Report Events with the JavaScript API

On this page:

The JavaScript API enables you to manually report events to the agent so that that it can time the various parts of your virtual page loads and correlate Ajax calls to those page loads. You can also capture and report errors using this API.

Notify the Agent of Events

Events are reported to the JavaScript Agent (ADRUM) by calling the `ADRUM.report` method and passing in an event tracker object.

API	Description
ADRUM.report(tracker: eventTracker);	Notifies the agent of an event.

Report Virtual Pages

SPA1 Monitoring

For SPA1 monitoring, use the event tracker VPageView to manually report virtual pages.

SPA2 Monitoring

You are required to enable SPA2 monitoring to use the API below to manually report virtual pages for SPAs. These APIs will also work in non-SPAs if you enable SPA2 monitoring.

To report virtual pages, you mark the beginning and end of virtual pages with the methods below. Both methods are called from the `ADRUM` object. See also Set Custom Virtual Page Names.

API	Parameter(s)	Default Value	Descriptions
`markVirtualPageBegin(VPName: string, manuallyMarkVPEnd?: boolean)`	`VPName`	N/A	Used to set the label for the virtual page. This label will be displayed in the Controller UI. The virtual page name must consist of a string of 760 or fewer alphanumeric characters. If the string length exceeds 760 characters, the page name will not be set.

	`manuallyMarkVPEnd`	`true`	A flag that indicates whether you or the JavaScript Agent mark the end of the virtual page. When set to true, you need to call `markVirtualPageEnd` to report the virtual page. When set to `false`, the JavaScript Agent will automatically mark the end of the virtual page.
`markVirtualPageEnd()`	N/A	N/A	Calling this method marks the end of the virtual page and triggers the JavaScript Agent to send a beacon with the virtual page information.

How the API Works

The steps below describe the process of manually reporting virtual pages with the API:

1. Start monitoring a virtual page by manually marking the beginning of the virtual page with `ADRUM.markVirtualPageBegin(VPName, manuallyMarkVPEnd)`.
2. A beacon with the set virtual page name is sent to the EUM Server. If `manuallyMarkVPEnd` is set to true, the JavaScript Agent will wait for you to call `ADRUM.markVirtualPageEnd` to report the virtual page. If `manuallyMarkVPEnd` is set to `false`, the JavaScript Agent will automatically mark the end of the virtual page.
3. You either call `ADRUM.markVirtualPageEnd()` to mark the end of the virtual page or the JavaScript Agent automatically marks the end of the virtual page.
4. The JavaScript Agent reports the virtual page metrics to the EUM Server.

Example of Reporting Virtual Pages

The Angular example below shows both ways to mark the beginning of a virtual page. The function `manualMarkVPEnd` calls `ADRUM.markVirtualPageBegin` that uses the default requiring you to manually mark the end of the virtual page. The function allowJSAgentMarkVPEnd passes the value `false` as the second parameter, so that the JavaScript Agent will automatically mark the end of the virtual page for you.

```
angular.module('myApp.controllers', [])
    .controller('VPCtrl', ['$scope', '$http', function ($scope, $http) {
            $scope.manualMarkVPEnd = function () {
                console.log("Mark the beginning of the virtual page and
wait for markVirtualPageEnd() to be called.");
                ADRUM.markVirtualPageBegin("VPExample-ManuallyMarkEnd");
            }
            $scope.allowJSAgentMarkVPEnd = function () {
                console.log("Mark the beginning of the virtual page and
allow the JS Agent to mark the virtual page end.");

ADRUM.markVirtualPageBegin("virtualPageExample-JSAgentMarksEnd", false);
            }
            $scope.endVirtualPage = function () {
                console.log("Mark the end of the virtual page.");
                ADRUM.markVirtualPageEnd();
            }
            ...
        }
    ]
);
```

Report Events

Events are reported to the agent using event trackers. There are three different kinds of event trackers:

Event Tracker	Enabled for SPA2 Monitoring?	Description
VPageView	No	Used to track the stages of a virtual page view.
Ajax	Yes	Used to track Ajax requests.
Error	Yes	Used to track errors.

Common Properties

There are also two properties that are common to all tracker types:

- Gets or sets a URL

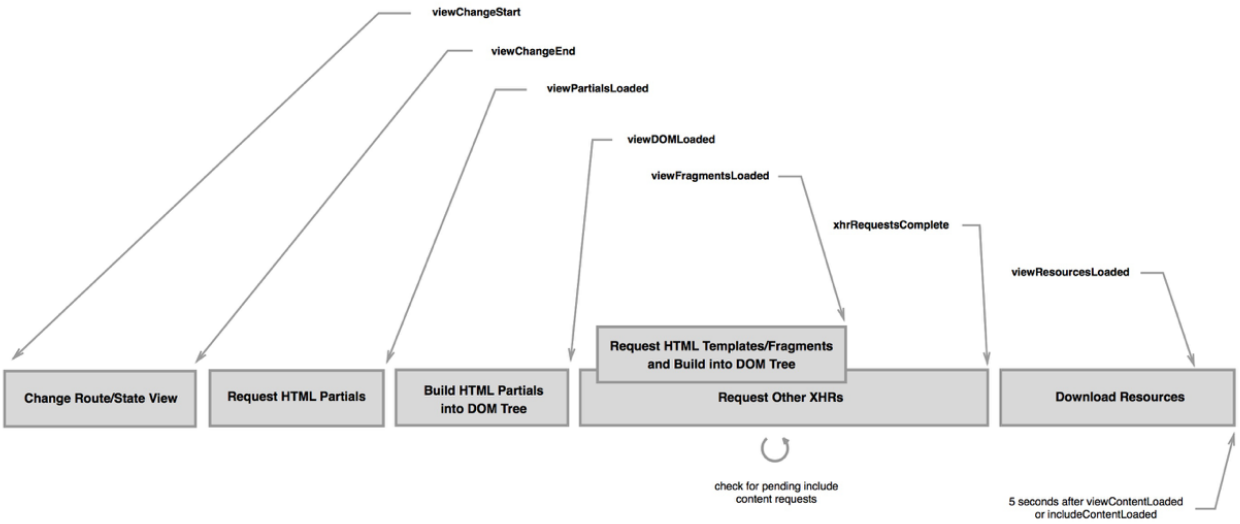
API	Description
`url(url?: string)`	Gets or sets a URL.

- Gets or sets the parent event identifier

API	Description
`parent(parent?: object)`	Gets or sets the parent event identifier.

VPageView

The following is the page view load flow in SPA1 monitoring. You'll use the SPA1 monitoring API below to set timing marks to match the below workflow as closely as possible in your own single page app framework. For SPA2 monitoring, see Report Virtual Pages: SPA2 Monitoring to learn about the SPA2 APIs for manually reporting virtual pages.



Based on the marks you set, AppDynamics derives the following key timing metrics. Marks should be called in the order in which they occur in the flow. The following table describes which marks used to calculate each metric.

Full Metric Name	Short Metric Name	How Calculated
End User Response Time (not used for waterfall UI)	PLT	`virtualPageStart` to `virtualPageEnd`
HTML Download Time	DDT	`viewChangeStart` to `viewChangeEnd`
HTML Download and DOM Building Time	DRT	`viewChangeStart` to `viewDOMLoaded`
DOM Building Time	DPT	`viewChangeEnd` to `viewDOMLoaded`
DOM Ready Time (used instead of PLT for waterfall UI)	DOM	`viewChangeStart` to `viewDOMLoaded`

Instantiate using `ADRUM.events.VPageView()`.

API	Description
`start()`	Indicates when a virtual page starts. It automatically calls: • `startCorrelatingXhrs()` • `markVirtualPageStart()`
`end()`	Indicates when a virtual page ends. It automatically calls: • `stopCorrelatingXhrs()` • `markVirtualPageEnd()`

`startCorrelatingXhrs()`	Correlates the Ajax requests sent after this call with the virtual page view event. The last tracker calling this method wins. This method is called automatically in the VPageView constructor. When a VPageView is created, the AJAX requests made after that call are automatically correlated to that VPageView. Use this separate call only when you wish to set up manual correlation.
`stopCorrelatingXhrs()`	Stops correlating Ajax requests to the virtual page view event. Use this separate call only when you wish to set up manual correlation.
Setters	The default value for these is the time when the API is called.
`markViewChangeStart()`	Sets the view change start time.
`markViewChangeEnd()`	Sets the view change end time.
`markViewDOMLoaded()`	Sets the view DOM loaded time.
`markXhrRequestsCompleted()`	Sets the XHR requests completed time.
`markViewResourcesLoaded()`	Sets the view resources loaded time. This includes images, CSS files, and scripts.
`markVirtualPageStart()`	Sets the virtual page start time.
`markVirtualPageEnd()`	Sets the virtual page end time.
Getters	
`getViewChangeStart()`	Gets the view change start time.
`getViewChangeEnd()`	Gets the view change end time.
`getViewDOMLoaded()`	Gets the view DOM loaded time.
`getXhrRequestsCompleted()`	Gets the XHR requests completed time.
`getViewResourcesLoaded()`	Gets the view resources loaded time.
`getVirtualPageStart()`	Gets the virtual page start time.
`getVirtualPageEnd()`	Gets the virtual page end time.

Ajax

Instantiate using `ADRUM.events.Ajax()`.

API	**Description**
Property Setters/Getters	Call this without a parameter to get the value and with a parameter to set the value.
`method(method?: string)`	Gets or sets the method ("GET" or "POST") of the Ajax.
Timing Setters	The default value for these is the time when the API is called.
`markSendTime(sendTime?: number)`	Sets the time the request is sent.

`markFirstByteTime(firstByteTime?: number)`	Sets First Byte Time.
`markRespAvailTime(respAvailTime?: number)`	Sets Response Available Time.
`markRespProcTime(RespProcTime?: number)`	Sets the time the response is completely processed.
Timing Getters	
`getSendTime()`	Gets the time the request was sent.
`getFirstByteTime()`	Gets First Byte Time.
`getRespAvailTime()`	Gets Response Available Time
`getRespProcTime()`	Gets the time the response was completely processed.

Errors

Instantiate using `ADRUM.events.Error()`.

API	Description
Property Setters/Getters	Call these without a parameter to get the value and with a parameter to set the value.
`msg(msg?: string)`	Gets or sets the error message.
`line(line?: number)`	Gets or sets the line number of source code where the error happened.

Correlate Ajax Requests

Ajax requests can be correlated to virtual page views automatically or manually. When you create a VPageView tracker, `startCorrelatingXhrs()` is called automatically in the constructor, correlating any subsequent Ajax calls with that VPageView event. To set up manual correlation, call `stopCorrelatingXhrs()` to stop the automatic process and then call `startCorrelatingXhrs()` where you wish correlation to re-commence.

Sample Code

Report a custom Error event by passing properties via setters

```
var errorT = new ADRUM.events.Error();
errorT.msg('I am a custom error at line 100');
errorT.line(100);
ADRUM.report(errorT);
```

Report a custom Error event by passing properties via the constructor

```
var errorT = new ADRUM.events.Error({
msg: 'I am a custom error at line 100',
line: 100
});
ADRUM.report(errorT);
```

Report a custom Ajax event passing properties via setters

```
var ajaxT = new ADRUM.events.Ajax();

// set url
ajaxT.url('your xhr Url');

// mark timings
ajaxT.markSendTime(100);
ajaxT.markFirstByteTime(200);
ajaxT.markRespAvailTime(300);
ajaxT.markRespProcTime(400);
ADRUM.report(ajaxT);
```

Set up backbone SPA monitoring

```
var AppRouter = Backbone.Router.extend({
    routes: {
        "wines/:id": "wineDetails"
    },
    wineDetails: function (id) {
        var vpView = new ADRUM.events.VPageView();
        vpView.markVirtualPageStart();
        // vpView.markViewChangeStart();
        var wine = new Wine({id: id});
        wine.fetch({success: function(){
            vpView.markXhrRequestsCompleted();
            $("#content").html(new WineView({model: wine}).el);
            vpView.markViewDOMLoaded();
            vpView.markVirtualPageEnd();
            ADRUM.report(vpView);
        }});
        this.headerView.selectMenuItem();
    }
});
```

Correlate Ajax requests wih VPageView Events

```
var vPageView = new ADRUM.events.VPageView({
    url: 'http://localhost/#virtualpage1',
});

vPageView.start();

// SPA view routing and HTML partials fetching
vPageView.markViewChangeStart()
// AJAX requests for the HTML partials are automatically correlated with
the VPageView
...
vPageView.markViewChangeEnd();

// HTML partials inserted into Browser DOM tree
...
vPageView.markViewDOMLoaded();

// SPA HTML AJAX data fetching
// Data AJAX requests are automatically correlated with the VPageView
...

vPageView.markXhrRequestsCompleted();

// call this when ending a new virtual page
vPageView.end();

ADRUM.report(vPageView);
```

You can exclude certain Ajax calls from being monitored by configuring ADRUM itself Before you invoke the `adrum.js` script at the top of your page, add lines similar to the following:

Exclude Ajax from VPageView using ADRUM configuration

```
window['adrum-config'] = {
  "spa": {
    "angular": {
      "vp": {
        "xhr": {
          "exclude": {
            "urls": [{
              "pattern": 'heartBeatAjax'
            }]
          }
        }
      }
    }
  }
}
```

Or you can exclude certain Ajax calls using the `vPageView.stopCorrelatingXhrs()` call, and then turn correlation back on with `vPageView.startCorrelatingXhrs()`, as in the following:

Exclude Ajax from VPageView event manually

```
var vPageView = new ADRUM.events.VPageView();
vPageView.stopCorrelatingXhrs();

var xhr = new XMLHttpRequest();
xhr.open('GET', '/heartBeatAjax');
xhr.send();

vPageView.startCorrelatingXhrs();
```