David Hertenstein

Core Team – OB Widget Documentation

8/21/2015

**OB Widget v1.0.0**

The OB Widget is a reusable pattern for injecting an Optimal Blue application widget on a customer page using plain web standards and a simplified interface for customer integration.

The example application includes the quick search functionality for Electronic Origination, or EO. In this example a customer wants to inject the Optimal Blue Widget into their web page to offer a quick quote search functionality. The customer site includes the OB Widget on an initial page as a simple search form. When the user selects their form entries and hits the submit button they are taken to a second page on the customer site containing a larger version of the same OB Widget. The Widget can open the second page in a new tab or within the same window based on the customer’s configuration.

The OB Widget is included on the customer page via an iframe that is injected dynamically when the page is loaded via a simple configuration call to the OB Widget Javascript library. The customer can define the widget’s width, height, border, and pass in a URL to their own CSS override file to style the widget as desired. The Widget can also be given custom values for the injection target element and the URL for the results page on the customer’s site.

To include the OB Widget on their site a customer needs to do the following in their host page source code on both an index page and a results page:

1. Include a script tag in the document HEAD pointing to the OB Widget injection library, hosted on Optimal Blue servers.
2. Include and inline script tag calling the OB Widget injection library with the customer’s desired arguments.
3. Include an html tag element, such as a DIV tag, with an ID that matches the ID argument used to inject the Widget.

Examples for the three requirements are as follows:

1. <script type="text/javascript" src="js/obWidget.1.0.0.js"></script>
2. <script type="text/javascript">

document.addEventListener('DOMContentLoaded', function (event) {

var cssUrl = OBWidget.locationStr() + '/css/external.css';

var resultsUrl = '/results.html';

OBWidget.initOBWidget('self', cssUrl, '350px', 1000px, 'none', 'OBWidget', resultsUrl, '//localhost:62900/api/search/show/1/2/3', '//localhost:62900/api/search/results', true, true);

});

</script>

1. <div id="OBWidget"></div>

**OB Widget Arguments**

The arguments for the OB Widget initOBWidget function that injects the widget into the customer page are as follows:

1. Self – (“self” or “tab”) A string defining if the results page should open in the same browser window or a new tab.
2. CSS URL – (“/css/external.css”) A string defining the relative path to a CSS file to inject into the OB Widget application for skinning the form and results elements. \*\*Additional documentation will be provided to map the skinnable elements.
3. Width – (“350px”) A string defining the CSS width of the iframe element.
4. Height – (“1000px”) A string defining the CSS height of the iframe element.
5. Border – (“none”) A string defining the CSS border of the iframe element.
6. Include ID – (“OBWidget”) A string defining the ID attribute of the html element to inject the OB Widget iframe into.
7. Results URL – (“/results.html”) A string defining the relative path to the results html page containing the expanded results view of the application.
8. Search API URL – (“//localhost:62900/api/search/show/1/2/3”) A string defining an API endpoint URL provided by Optimal Blue for the search page.
9. Results API URL – (“//localhost:62900/api/search/results”) A string defining an API endpoint URL provided by Optimal Blue for the results page.
10. Search flag – (true) A Boolean defining whether the page is the search page on the customer website, false is used for the results page.
11. Bootstrap CSS styling – (true) A Boolean defining whether the page uses the bootstrap framework or not for its default styling.

**CSS Skinning**

The OB Widget takes an argument that is the last variable that toggles the forms to use the Twitter Bootstrap framework or to be plain, html elements with basic css styles. The second argument can also be used to pass in the relative URL of a CSS file that the widget will include as an external style sheet for custom skinning. Using this external style sheet the customer can skin and style either the default bootstrap styles or the generic unstyled css used by the OB Widget. To skin the bootstrap styles the customer should refer to the bootstrap version 3 documentation, found here:

<http://getbootstrap.com/css/>

If the customer opts to use the generic html version of the OB Widget, they can skin the form elements using the following set of general css classes and selectors built into the non-bootstrap OB Widget.

**Generic CSS styles:**

/\* ===GLOBAL=== \*/

.widthFull {

width: 100%;

}

.width98 {

width: 98%;

}

.widthTwoThirds {

width: 66%;

}

.widthQuarter {

display: inline-block;

width: 24%;

vertical-align: top;

}

/\* ===Panel=== \*/

#plainPanel {

border: 1px solid #F0F0F0;

}

#plainPanel #heading {

padding: .3em;

background-color: #F0F0F0;

}

/\* ===Tabel=== \*/

#plain-table {

}

table#plain-table thead {

}

table#plain-table thead tr {

}

table#plain-table thead tr th {

}

table#plain-table tbody {

}

table#plain-table tbody tr {

}

table#plain-table tbody tr td {

}

/\* ===Inputs/Buttons=== \*/

#inputs-padding {

padding: .3em;

}

#input-wrapper {

width: 100%;

}

#input-label {

width: 33%;

}

#input-box {

width: 66%;

}

#select-wrapper {

width: 100%;

}

#select-label {

width: 33%;

}

#select-box {

width: 66%;

}

#submit-button {

}

/\* ===Accordion=== \*/

.accordion-container {

width: 100%;

}

.expandcollapse-item {

border-top: 1px solid #F0F0F0;

overflow-y: auto;

}

.expandcollapse-heading-collapsed {

cursor: pointer;

padding: .3em;

color: black;

background-color: #F0F0F0;

margin-bottom: 1em;

}

.expandcollapse-item:first-of-type {

border-top: 0;

}

.expandcollapse-heading-collapsed p {

margin: 0;

}

.expandcollapse-heading-expanded {

cursor: pointer;

padding: .3em;

color: black;

background-color: #F0F0F0;

border: 1px solid #F0F0F0;

}

.expandcollapse-heading-expanded p {

margin: 0;

}

.expand-collapse-content {

padding: .3em;

border-left: 1px solid #F0F0F0;

border-right: 1px solid #F0F0F0;

border-bottom: 1px solid #F0F0F0;

margin-bottom: 1em;

}

/\*animation:\*/

.slideDown.ng-hide {

height: 0;

-moz-transition: height 0.35s ease;

-o-transition: height 0.35s ease;

-webkit-transition: height 0.35s ease;

transition: height 0.35s ease;

overflow: hidden;

}

.slideDown {

-moz-transition: 0.35s ease;

-o-transition: 0.35s ease;

-webkit-transition: 0.35s ease;

transition: 0.35s ease;

overflow: auto;

}

.slideDown.ng-hide-remove,

.slideDown.ng-hide-add {

/\* remember, the .hg-hide class is added to element

when the active class is added causing it to appear

as hidden. Therefore set the styling to display=block

so that the hide animation is visible \*/

display: block !important;

}

.slideDown.ng-hide-add {

-moz-animation-name: hide;

-o-animation-name: hide;

-webkit-animation-name: hide;

animation-name: hide;

-moz-animation-duration: .5s;

-o-animation-duration: .5s;

-webkit-animation-duration: .5s;

animation-duration: .5s;

-webkit-animation-timing-function: ease-in;

-moz-animation-timing-function: ease-in;

-o-animation-timing-function: ease-in;

animation-timing-function: ease-in;

}

.slideDown.ng-hide-remove {

-webkit-animation-name: show;

-moz-animation-name: show;

-o-animation-name: show;

animation-name: show;

-webkit-animation-duration: .5s;

-moz-animation-duration: .5s;

-o-animation-duration: .5s;

animation-duration: .5s;

-webkit-animation-timing-function: ease-out;

-moz-animation-timing-function: ease-out;

-o-animation-timing-function: ease-out;

animation-timing-function: ease-out;

}