**Responsive iFrame API Documentation**

*Updated on 3/19/15*

*1. Changed definition of “addPageListener” and “removePageListener” to reflect new version that now takes the callback parameter as a function instead of a string.*

*2. Added “getPubPageInfo” function to the adside.js giving the user the ability to access info about the publisher page like window width/height without adding a listener.*

**Introduction**

This API is an adaptation of the SafeFrames initiative, allowing ads to be responsive while served inside an iFrame. There are two main parts of this process.. the pub-side file, and the ad-side code. The pub-side file lives on the publisher page and acts as a bridge to the ad-side code that posts messages and directions to be carried out.

**Requirements**

1. The publisher must have the resp.lib.js file included on their page prior to the ad being served.
2. The creative must have the API class included in order to initialize and use the Responsive iFrame API.
3. The placements must define the variable “prPlacementId” to the placement id. This is imperative so that the API has a reference to which placement should be utilized, especially if there’s more than one PointRoll ad on the page. This is the id that is sent to the initialization function of the API.

**Known Limitations**

1. The ad must be in an iFrame. The iFrame can be PointRoll *or* site generated.
2. If the iFrame is not PointRoll generated, the tag may not be altered to use the ads.pointroll call as the source. This will result in a double nested iFrames and the API will not work.

**Library Definition**

In order to communicate out to the publisher page, you must include the prAdConnection class within the creative. This serves as the bridge between the ad and the publisher page by sending and receiving calls.

**Initialization**

Create an instance of the ad side library by setting a reference variable to the class. This accepts an optional boolean parameter that can be used to enable the debug mode which will show console logs in a browser’s web inspector. If no boolean is passed, it will default to false.

*Example:*

var \_pr\_ad\_connection = new prAdConnection(true);

**createConnection** - function(*ad wrapper id, break points, id, ad slot index*)

When the ad is ready, make this call to create the connection between the pub-side file and the ad.

*Parameters:*

ad wrapper id (string)

ID of the ad’s outer most div.

break points (object)

An object of break points the ad intends to use. This object should be comprised of a “width” and “height” key with values of type array with the break points inside.

id (number or string)

The PointRoll placement ID associated with the ad.1

ad slot index (number or string)

A reference to the number of divs the iFrame is nested in on the publisher page.2

*Example:*

var \_pr\_responsive\_break\_points = {‘width’: [468, 728, 2014], ‘height’: []};

\_pr\_ad\_connection.createConnection(document.getElementById(‘pr-wrap’, \_pr\_responsive\_break\_points, 217382, 3);

**Other Ad-Side Functions**

**addPageListener** - function(*target, type, callback*)

Add listeners to the publisher page.

*Parameters:*

target (string)

This is the element you are adding the listener to. Should be either “window”, “document”, or a specific element ID.

type (string)

The listener in which you are adding to the target i.e. “mouseover”, “scroll”, or “click”.

callback (function)

The listener’s handler function that should be called when the listener is fired. See appendix for details on the returned event object.3

*Example:*

\_pr\_ad\_connection.addPageListener(‘window’, ‘scroll’, windowScrollHandler);

\_pr\_ad\_connection.addPageListener(‘window’, ‘resize’, function(event){

var \_width = event.innerWidth;

});

function windowScrollHandler(event){

var \_scroll = event.scrollY;

}

**closeAd** – function(*no parameters*)

Completely removes the ad from the page and destroys the connection.

*Example:*

\_pr\_ad\_connection.closeAd();

**getAdSlotSize** – function(*no parameters*) -> *read only*

Returns an object containing the ad slot width and height. \*Please note, due to the need for the ad-side API to talk to the pub-side API, this call must be placed on a small timeout in order to ensue the correct values are received.

*Example:*

var \_pr\_slot\_width = \_pr\_ad\_connection.getAdSlotSize().width;

**getBrowserInfo** – function(*no parameters*) -> *read only*

Returns an object containing the browser type, browser version, device type, and device version.

*Example:*

\_pr\_ad\_connection.getBrowserInfo();

**getPubPageInfo** – function(*no parameters*) -> *read only*

Returns an object containing the window width/height, body width/height, window scroll, and iframe boundingRect.

*Example:*

var \_window\_width = \_pr\_ad\_connection.getPubPageInfo().window.width;

**log** - function(*message*)

Use this instead of console.log in order to only fire a console log if the debug mode has been set to true.

*Parameters:*

message (string)

Message to be logged.

*Example:*

\_pr\_ad\_connection.log(‘function has completed’);

**removePageListener** - function(*target, type, callback*)

Removes listeners from the publisher page.

*Parameters:*

target (string)

This is the element you are adding the listener to. Should be either “window”, “document”, or a specific element ID.

type (string)

The listener in which you are adding to the target i.e. “mouseover”, “scroll”, or “click”.

callback (function)

The listener’s handler function that should be called when the listener is fired. See appendix for details on the returned event object.3

*Example:*

\_pr\_ad\_connection.removePageListener(‘window’, ‘scroll’, windowScrollHandler);

\_pr\_ad\_connection.removePageListener(‘window’, ‘resize’, function(event){

var \_width = event.innerWidth;

});

**Receiving Pub-Side Messages**

**updateAdSize** – callback function(*size, type*)

Once the connection is made between the ad and the publisher page, the pub-side file will begin broadcasting the browser width and height to the iFrame. The ad-side API object will determine if a new break point has been reached and then fire a callback with the new size. This is where the ad should handle the responsive portion of the creative build. With the updated size comes the typeparameter that determines whether the new size is in regards to the browser width or height. Use a switch-case statement to make your creative changes based on the current ad size. Define this callback with your creative script code.

*Parameters:*

size (number)

The size will be returned in the form of a number index correlating to the break points you define.

type (string)

This will return whether the size callback is for “width” or “height”.

*Example:*

\_pr\_ad\_connection.updateAdSize = function(size, type){

switch(type){

case ‘width’:

switch(size){

case 0:

\_pr\_ad\_connection.sendMessageToPub({‘setAdSize’: {‘width’: ‘100%’, ‘height’: ‘90px’}});

break;

case 1:

\_pr\_ad\_connection.sendMessageToPub({‘setAdSize’: {‘width’: ‘100%’, ‘height’: ‘250px’}});

break;

}

}

}

}

**fireNonImpression** – callback function(*no parameters*)

If the ad is hidden on the initial view, this callback function will fire off to give the ad the opportunity to either track or react to this event in other ways. This callback will not fire off if the ad is initially viewed and then hidden, or on subsequent hides.

*Example:*

\_pr\_ad\_connection.fireNonImpression = function(){

//fire non-impression pointroll activity here

}

**Sending Messages To The Pub-Side File**

**sendMessageToPub** – function(*message*)

This is the call used to send messages / directions to the pub-side file in order to alter the iFrame, etc.

*Parameters:*

message (object)

The message is compiled of the pub-side call and the parameters associated with it. Since this is an object, you can send more than one message to the pub in one line.

*Example:*

\_pr\_ad\_connection.sendMessageToPub({‘setAdSize’: ‘100%,250’, ‘addCustomStyles’: ‘margin=20px 0px 20px 0px;’});

**Available Messages**

**addCustomStyles** – (*styles*)

Add styles to the iFrame or parent nodes. Create a JSON object containing all style properties and their corresponding values.

*Parameters:*

styles (object)

Each style key must have a corresponding style value.

*Example:*

\_pr\_ad\_connection.sendMessageToPub({‘addCustomStyles’: {‘margin-top’: ‘50px’, ‘margin-bottom’: ‘50px’}});

**addDirectStyles** – (*target and styles*)

Add styles to any specified class(es) or id(s). Create a JSON object containing all elements and their corresponding styles.

*Parameters:*

target and styles (object)

Use “#your\_id” or “.your\_class” and separate with a “,” if more than one element is going to receive the properties.

*Example:*

var \_pr\_options = {

‘.test-class1, .test-class2’: {

‘margin’: ‘0px’

},

‘#test-id1’: {

‘background’: ‘#FFF’

}

};

\_pr\_ad\_connection.sendMessageToPub({‘addDirectStyles’: + \_pr\_options});

**destroy** – (*no parameters*)

Removes all API window listeners on the publisher page.

*Example:*

\_pr\_ad\_connection.sendMessageToPub({‘destroy’: null});

**hide** – (*no parameters*)

Hides the ad and collapses the ad slot without removing the ad from the page.

*Example:*

\_pr\_ad\_connection.sendMessageToPub({‘hide’: null});

**overwriteImportantStyles** – (*target and styles*)

Same implementation as the “addDirectStyles” function but should be used only if the target has !important implemented on the styles you wish to overwrite.

*Parameters:*

target and styles (object)

Use “#your\_id” or “.your\_class” and separate with a “,” if more than one element is going to receive the properties.

*Example:*

var \_pr\_options = {

“.test-class1, .test-class2”: {

“margin”: “0px”

},

“#test-id1”: {

“background”: “#FFF”

}

};

\_pr\_ad\_connection.sendMessageToPub({‘overwriteImportantStyles’: + \_pr\_options});

**reveal** – (*no parameters*)

Shows the ad and un-collapses the ad slot.

*Example:*

\_pr\_ad\_connection.sendMessageToPub({‘reveal’: null});

**setAdOffset** – (*x, y*)

Offset the top and left of the iFrame.

*Parameters:*

x and y (object)

X and y offset position for the ad slot. Can also accept percentages.

*Example:*

\_pr\_ad\_connection.sendMessageToPub({‘setAdOffset’: {‘x’: ‘10px, ‘y’: ‘20px’}});

**setAdPosition** – (*position*)

Change the positioning of the iFrame.

*Parameters:*

position (string)

Standard positioning style of absolute, fixed, relative, etc.

*Example:*

\_pr\_ad\_connection.sendMessageToPub({‘setAdPosition’: ‘absolute’});

**setAdSize** – (*width, height*)

Change the size of the iFrame.

*Parameters:*

Width and height (object)

Width and height of the ad slot. Can also accept percentages.

*Example:*

\_pr\_ad\_connection.sendMessageToPub({‘setAdSize’: {‘width’: ‘100%’, ‘height’: ‘250px’}});

**setAdSlotTarget** – (*index*)

Change the index of the targeted ad slot.

*Parameters:*

index (number)

Index of the ad slot.

*Example:*

\_pr\_ad\_connection.sendMessageToPub({‘setAdSlotTarget’: 2});

**Appendix**

1. There is no PointRoll macro associate with this ID so the best way to obtain this would be to either hard code it in the main ad script file or set it as a variable at the placement level and access that variable.
2. When the ad is served to the page, it’s not necessarily served as the next child of the ad slot. Sometimes it can be nested within multiple divs. In most cases, this means that applying styles to just the parent node of our iFrame may not be enough. For example, if we are in a div that has a parent node with a height style set, we can’t just set the height of the div we’re inside. By setting the ad slot index to 2, the API now knows which parent node to access.
3. All callback functions will be sent a copy of the event object returned at the pub level. For listeners like window scroll and window resize, the information needed cannot be found in the standard event. Window scroll handlers usually use window.scrollX/Y and window resize usually uses window.innerWidth/Height. For these two examples specifically, the scrollX/Y and innerWidth/Height can be accessed via the event parameter sent to the callback function. For example, instead of using window.scrollY, use event.scrollY. Also, all events will get returned a “rect” key that corresponds to the iFrame getBoundingClientRect property.