Integrating Third-Party Device Recognition JavaScript

Overview

These topics walk you through integrating the iovation third-party JavaScript into your web pages. The iovation JavaScript collects device information that you can then submit to iovation as part of your Customer Authentication or Fraud Prevention implementation. To ensure the best device recognition results for web pages, iovation integration incorporates two types of JavaScript, first-party and third-party, each with its own configuration steps. This comprises our required integration approach for web sites, *Multi-Domain Recognition*.

Multi-Domain Recognition is *Required* for All Web Integrations

To successfully integrate iovation **you must integrate both first-party and third-party JavaScript into your web pages. Both first-party and third-party integration are required.**

Setting Up Third-Party Integration on Web Pages

IMPORTANT! Do not cache or use local copies of snare.js. The iovation JavaScript is dynamically generated for every user; caching the script may cause unrelated devices to be identified as the same computer. The script also uses domain cookies to identify devices across subscribers.

Including the Script

You must include the third-party script on your web pages. Current version information and URLs for test and production versions of the third-party script is listed below.

File name	snare.js
URL	• Production: https://mpsnare.iesnare.com/snare.js • Test: https://ci-mpsnare.iovation.com/snare.js
Version	3.1.0.0
File size	~31 KB (compressed to ~12 KB)

To include the third-party iovation script, add the following `<script>` element to your web page:

```
<script type="text/javascript" src="https://mpsnare.iesnare.com/snare.js"></script>
```

NOTE Note that this is the URL to the version of snare.js in the iovation production environment.

Configuring Third-Party JavaScript

These variables determine how the iovation script collects data and interacts with your end-users.

IMPORTANT! You must place the configuration parameters before the call to the iovation script. The configuration parameters determine how the iovation script will operate. Errors can result if these parameters are not placed before the call to the iovation script.

Add the following parameters to a `<script>` on your page. These should follow standard HTML script notation. For example:

```
<script>
  var io_install_flash = false;
  var io_install_stm = false;
</script>
```

Configuration Value	Type	Default Value	Description
`io_install_flash`	boolean, optional	`false`	Determines whether the user is prompted to install or upgrade Flash if the required version of Flash is not installed. If the required version of Flash is installed, this setting has no effect.
`io_flash_needs_handler`	string		If `io_install_flash` is set to false, this handler will not run. Users may see an error if the required version of Flash is not installed to their systems. Use this variable to define your own JavaScript error handling for this condition. For example, you can define your own error message: `var io_flash_needs_update_handler = "alert( 'Install Flash!' );";`
`io_install_stm`	boolean, optional	`false`	Determines whether the user is prompted to install the iovation Active X control. The Active X control helps collect hardware information. This control is only available for Internet Explorer. If the Active X control is installed, this setting has no effect.
`io_exclude_stm`	bitmask, optional	`15`	Determines whether to use the iovation Active X control, if it is installed. When Active X is available in Internet Explorer (version 8+ and on Windows Vista), security warnings may appear when the control is installed but not yet run. You can opt to disable the control for specific platforms and therefore prevent the prompt from occurring.

Configuration Value	Type	Default Value	Description
			This bitmask has the following set of values: • `0` – default (run on all platforms) • `1` – Do not run on Windows 9x (including Windows 3.1, 95, 98 and Me) • `2` – Do not run on Windows CE • `4` – Do not run on Windows XP (including Windows NT, 2000, 2003, and 8) • `8` – Do not run on Vista or newer versions of Vista • The values are additive, so a value of `12` means do not run on either Vista (`8`) or Windows XP (`4`).
`io_bbout_element_id`	string, optional	`none`	The ID of the HTML element to populate with the blackbox from the third-party JavaScript. If `io_bb_callback` is specified, this parameter has no effect.
`io_enable_rip`	boolean, optional	`true`	Indicates whether iovation will attempt to collect information to determine the Real IP of the end user.
`io_bb_callback`	function, optional		This JavaScript function is an event handler that is called when a collection method has finished updating a blackbox. Declare the function as follows: `function io_callback( bb, complete)` The variables store the following: • `bb` – the updated value of the blackbox • `complete` – a boolean value indicating whether all the collection methods have completed.

Collecting a Blackbox

Methods for Collecting Third-Party Blackboxes

The iovation scripts create two blackboxes, one using the first-party scripts and one using the third-party script. After the scripts create blackboxes, you must collect and send them back to your server. Your server will then combine the blackboxes and send them to iovation to process.

There are three ways to collect a blackbox.

- Populate a hidden form field using the `io_bbout_element_id` variable.
- Define the `io_bb_callback` function to collect the blackbox as it is generated.
- Call the JavaScript function `ioGetBlackbox` to obtain the blackbox. You can combine `ioGetBlackbox` and either of the other two methods.

Implementing the Hidden Form Field Collection Method

You can define a hidden form field that the iovation script, snare.js, populates with the blackbox. The blackbox is then submitted along with other fields in the form. To use this method you must define `io_bbout_element_id`. This method is primarily useful when you have a single form. For other cases, consider using one (or both) of the other methods.

1. Add a hidden field to a form on your web page and set the `id` attribute for the field. This field will store the blackbox. Give the `id` attribute a descriptive name; you will need to reuse this later. Here is an example of a simple form with a hidden field, with the `id` attribute set to `ioBlackBox`:

```
<form action="/do_ctd" method="post" name="loginSubmitForm" id="loginSubmitForm">
            <fieldset>
                <ul>
                    <!-- Create hidden for blackboxes to go into -->
                    <input TYPE="hidden" NAME="ioBlackBox" id="ioBlackBox">
                    <li><input type="submit" value="Do CTD" id="submit1" name="submit1">
                    </li>
                </ul>
            </fieldset>
        </form>
```

2. In the JavaScript configuration parameters, set the `io_bbout_element_id` parameter to the id of the hidden form field. For example, if the `id` attribute is set to `ioBlackBox`, set `io_bbout_element_id` to the following:
`var io_bbout_element_id = 'ioBlackBox';`

Implementing the Callback Function Collection Method

The callback interface allows you to manage blackbox generation in a more event-driven manner. As blackbox collection progresses, snare.js fires update events when a collection method has completed. These events trigger a user-defined callback function to update the page with the new blackbox value. When all of the collection methods are completed, a Boolean flag is set indicating no further updates are expected and the value is the final blackbox value.

1. In the JavaScript configuration parameters, set the `io_bb_callback` parameter to a function that processes the blackbox value, and that has the following signature:
`function bb_update_callback( bb, complete)`
Where:
    - `bb` is the updated value of the blackbox

- `complete` is a boolean value that indicates whether all collection methods (Flash, Active X etc.) are complete

  NOTE If `io_bb_callback` and `io_bbout_element_id` are both specified, the hidden field specified in `io_bbout_element_id` will not be populated, unless explicitly done so by the function specified in `io_bb_callback`.

Implementing the ioGetBlackbox Collection Method

The last method for obtaining a blackbox is to use the `ioGetBlackbox` function. This function returns an object that contains the current value of the blackbox along with a flag indicating whether the collection process has completed. This method is useful when the value is needed after the collection process has completed. It is also useful as a way to obtain the best value after some maximum amount of time to avoid any further delays in the user experience.

To implement the `ioGetBlackBox` collection method:

1. Call the function. For example:

   ```
   var blackbox_info = ioGetBlackbox();
   ```

   This returns an object with the following attributes:

   - `blackbox` – the updated value of the blackbox
   - `finished` – a Boolean indicating whether all the collection methods have completed.

Troubleshooting Errors

If you encounter errors such as failing to get a blackbox, you can use the `io_last_error` variable to troubleshoot. This variable stores the last error encountered in the third-party JavaScript. Check the value of this variable to look for errors caught while processing the third-party JavaScript.

The following example is a global JavaScript variable; you can review it with any JavaScript that runs after the iovation JavaScripts have loaded. For example, use the following snippet to store error information from both first-party and thirty-party integration:

```
<script>
console.log( "Last errors: [" + io_last_error + "] first party error: [" + fp_last_error + "]");
</script>
```

Next Steps

To successfully integrate iovation you must integrate both first-party and third-party JavaScript into your web pages. **Both first-party and third-party integration are required. If you have not already integrated first-party JavaScript, you must do so.**

Blackbox Collection Examples

Overview

These examples will help you integrate iovation risk services into your web pages.

Hidden Form Field Collection Example

Here is a sample web page with full integration code in place for both first-party and third-party scripts, and that uses the hidden form field collection method.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Demo Integration Page</title>
        <meta charset="UTF-8">
    </head>

<body>
    <div id="main">
        <form action="/do_ctd" method="post" name="loginSubmitForm" id="loginSubmitForm">
            <fieldset>
                <ul>
                   <!-- Create two hidden fields for the blackboxes to go into, one for the blackbox
generated by the 3rd party JavaScript, and another for the blackbox generated by the 1st party
JavaScript. In this example, the 1st party blackbox is named ioBlackBox, and the 3rd party blackbox
is named fpBB. -->
                <input TYPE="hidden" NAME="ioBlackBox" id="ioBlackBox">
                <input TYPE="hidden" NAME="fpBB" id="fpBB">
                <li><input type="submit" value="Do a CTD" id="submit1" name="submit1"
tabindex="4"></li>
                </ul>
            </fieldset>
        </form>
    </div>
    <!-- These variables execute before the JavaScript files load. They set default values for the
JavaScript. -->
    <script>
    // Variables for iovation hosted JavaScript
        var io_bbout_element_id          = 'ioBlackBox';
        var io_install_stm               = false;
        var io_exclude_stm               = 12;
        var io_install_flash             = false;
        var io_enable_rip                = true;
    // Variable for first party JavaScript
        var fp_bbout_element_id          = 'fpBB';
    </script>

<!-- Include the third-party iovation JavaScripts -->
<script language="javascript" src="https://ci-mpsnare.iovation.com/snare.js"></script>
<!-- Include first-party JavaScripts -->
<!-- IMPORTANT: Include the full path to static_wdp and dyn_wdp to prevent errors with Internet
Explorer  -->
<script language="javascript" src="static_wdp.js"></script>
<script language="javascript" src="/iojs/4.1.6/dyn_wdp.js"></script>
</script>
</body>
</html>
```

Callback Collection Example

The following example illustrates using the callbacks from each set of scripts to update multiple form fields. This simulates the case where an end user is reviewing an order that has a submission form at the top and bottom of the page. The example updates all fields when there is an update - not simply when collection is finished. This ensures that iovation will have something to work with, even when both collection methods do not fully complete.

```
<html>
<body>
<!-- Form 1 -->
  <form method=POST action="#">
    <input type=hidden name=fp_bb></input>
    <input type=hidden name=io_bb></input>
    <input type=submit name="Go first!"></input>
  </form>
<!-- Form 2 -->
  <form method=POST action="#">
    <input type=hidden name=fp_bb></input>
    <input type=hidden name=io_bb></input>
    <input type=submit name="Go second!"></input>
  </form>
  <script type="text/javascript">
     // standard config variables
     var io_install_flash = false;
     var io_install_stm = false;
     var io_exclude_stm = 12;
     var io_enable_rip = true;
     // third party callback function
     var io_bb_callback = function (bb, complete) {
         var fields = document.getElementsByName( "io_bb" );
         var i = 0;
         for ( i = 0; i < fields.length; i++ )
            fields[i].value=bb;
     };
     // first party callback function
     var fp_bb_callback = function (bb, complete) {
         var fields = document.getElementsByName( "fp_bb" );
         var i = 0;
         for ( i = 0; i < fields.length; i++ )
            fields[i].value=bb;
     };
   </script>
   <!-- // include scripts
       Third-party script
    -->
   <script type="text/javascript" src="https://ci-mpsnare.iovation.com/snare.js"></script>
   <!--
       First-party scripts; URLs here are to iovation test environments, actual URLs depend on
first-party implementation
    -->
   <script type="text/javascript" src="static_wdp.js"></script>
   <script type="text/javascript" src="/iojs/4.1.6/dyn_wdp.js"></script>
</body>
</html>
```

Get Blackbox Functions Collection Example

This example illustrates how to use the blackbox methods (`fpGetBlackbox` for first-party and `ioGetBlackbox` for third-party). Submission of the form initates the capture of the blackboxes. You can alter the send function to post the blackboxes via AJAX or perform some other operation. In the example, an alert displays the collected values.

```
<html>
<body>
  <form method=POST action="#">
    <input type=submit name="Go first!" onclick="return send_bb();"></input>
  </form>
  <script type="text/javascript">
     var io_install_flash = false;
     var io_install_stm = false;
     var io_exclude_stm = 12;
     var io_enable_rip = true;
     function send_bb() {
         // make AJAX call here or do something else with blackbox
         // for illustration purposes, we are just going to do an alert here
         var bb1 = fpGetBlackbox();
         var bb3 = ioGetBlackbox();
         alert( "First party bb: " + bb1.blackbox );
         alert( "Third party bb: " + bb3.blackbox );
     }
   </script>
   <!-- // include scripts
       third-party script
    -->
   <script type="text/javascript" src="https://ci-mpsnare.iovation.com/snare.js"></script>
   <!--
       First-party scripts; URLs here are to iovation test environments, actual URLs depend on
first-party implementation
    -->
   <script type="text/javascript" src="static_wdp.js"></script>
   <script type="text/javascript" src="/iojs/4.1.6/dyn_wdp.js"></script>
</body>
</html>
```

What's Next?

If you have finished implementing blackbox collection, continue to implementing your back-end server in order to send transactions to iovation. See Sending Transactions to iovation.