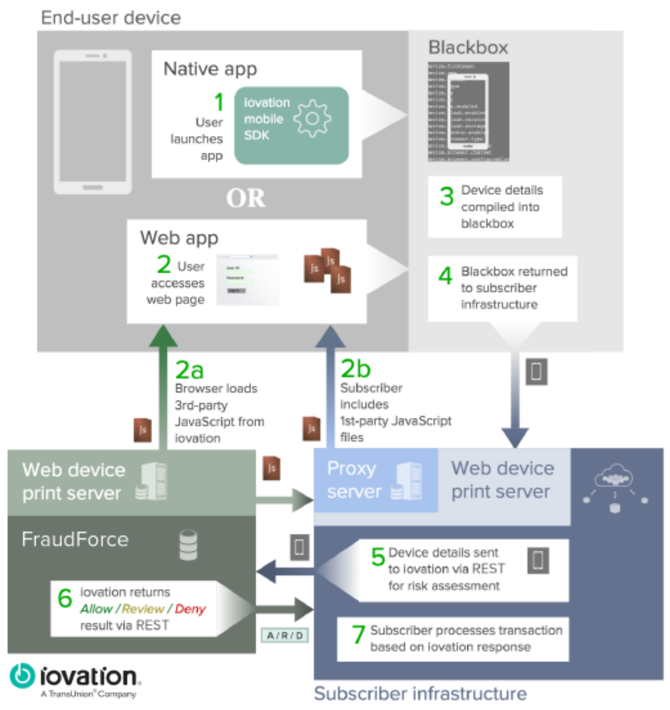
FraudForce RESTful Integration Guide

Integration Workflow

Overview

This is a workflow for iovation FraudForce using web pages and/or apps that embed the iovation SDK.

FraudForce Workflow


End-user device
Native app
1 User launches app
iovation mobile SDK
OR
Web app
2 User accesses web page
Blackbox
3 Device details compiled into blackbox
4 Blackbox returned to subscriber infrastructure
2a Browser loads 3rd-party JavaScript from iovation
2b Subscriber includes 1st-party JavaScript files
Web device print server
FraudForce
6 iovation returns Allow / Review / Deny result via REST
Proxy server
Web device print server
5 Device details sent to iovation via REST for risk assessment
7 Subscriber processes transaction based on iovation response
A/R/D
iovation
A TransUnion Company
Subscriber infrastructure

Step	Description
1	The end user either loads an app (mobile or desktop) that is integrated with iovation.
2	Or, opens a web page that is integrated with iovation's JavaScript. iovation's Javascript will load:

Step	Description
	• 2a: Third-party JavaScript from iovation and enable sharing of information. • 2b: First-party dynamic JavaScript from your web servers.
3	Either the embedded SDK or the web integration JavaScript builds a blackbox(es) that contains all device details.
4	The user submits their information to the subscriber's web server which includes the blackbox. For web integrations, the blackboxes are typically sent through hidden form fields on the page.
5	Using the CheckTransactionDetails API, your web server sends the blackbox(es) to iovation to evaluate, along with: • A unique transaction identifier • The name of the rule set to use to process the transaction • For web integrations, the user's stated IP address
6	iovation evaluates the transaction and device characteristics based on the business rules in the rule set. If the device and / or account are already known, iovation checks for any history of confirmed fraud and abuse. Based on this risk assessment, iovation returns a recommendation to allow, review, or deny the transaction.
7	The subscriber processes the transaction accordingly, accordingly to internal workflows.

How Web Integration Works

What is Multi-Domain Recognition?

iovation device recognition employs scripts from both your domain (the First-party domain) and iovation's domain (the Third-party domain). Including scripts from both of these domains allows iovation to:

- **Third-Party JavaScript:** Share fraud history for devices and accounts across iovation subscribers.
- **First-Party JavaScript:** Collect device information for users whose browsers are configured to disable third-party JavaScript, or that block the iovation domain.

iovation will provide a script for you to include on your site. This script will load the appropriate resources from both sets of domains. Components loaded by the script are dynamically generated and therefore not included with the script provided, nor should they be directly included on your page.

Retrieving the dynamic first party script will require some set up on your servers or network.

First-Party Dynamic JavaScript Options

Retrieving the dynamic first-party script can be done by:

- Setting up a reverse proxy to get the files from an iovation-hosted server
- Installing your own Java device print server.

Retrieving First-Party Dynamic JavaScript

Overview

You must serve up the iovation first-party dynamic JavaScript from within your own infraustructure. You can do this in one of the following ways:

- Use a reverse proxy to get the JavaScript components from iovation
- Host a WAR file, provided by iovation, that includes the JavaScript components

Retrieving First-Party Dynamic JavaScript Components via a Reverse Proxy

About Reverse Proxy Configuration

To deliver the first-party dynamic JavaScript components:

- You must tunnel requests through your site to iovation's servers.
- To do this, set up a reverse proxy that will forward all requests sent to a specific URI on your site to iovation's servers. You can define this URI in the configuration variables for the iovation-provided script.

Test and Production Server URLs

iovation provides the first-party dynamic JavaScript components from both test and production servers, at the following URLs:

Test	https://ci-first.iovation.com/*
Production	https://first.iovation.com/*

Do Not Cache Dynamic Components Using Content Distribution Networks (CDNs)

When using a Content Distribution Network (CDN) such as Akamai, ***make sure that the dynamic files are not cached by the CDN***. Caching these files results in sets of dynamic attributes becoming fixed for disparate sets of users, severely disrupting the recognition process. can severely disrupt the recognition process as the dynamic attributes will become fixed for a large disparate set of customers.

For Akamai, ensure that the caching option for the reverse proxy rule is set to no-store. This will prevent Akamai from caching the content and instead get new copies each time.

Configuring the Reverse Proxy

Important Configuration Notes

When setting up the reverse proxy:

- Because multiple resources may be requested, you must limit which specific files are forwarded.
- However, also provide a generic-enough mapping so that when iovation adds or updates resources in the future, you will get all of the updates.

IMPORTANT! When setting up a reverse proxy, ***do not:***

- Forward requests to the URI directly to iovation through a redirect; these requests can be blocked
- Create a special sub-domain; this can be just as easily blocked as iovation's domain
- Create a custom handler that intercepts the request and then requests the content on behalf of the customer; this prevents various HTTP headers from being analyzed from the user's machine. Standard reverse proxies will preserve those headers.

Configuration Steps

To configure the reverse proxy:

1. Set up a proxy configuration within your domain.
2. Specify a URI on your site to proxy the request. The default URI is `iojs`, however you can change this in the configuration section of your iovation JavaScript. Using the default URI, you would forward to iovation any request beginning with: http://my.domain.com/iojs
3. Direct the proxy to forward the requests to the following URL: https://first.iovation.com

Example: Configuring a Reverse Proxy in Nginx

Edit the Nginx configuration file (**default.conf**) and add the following lines to the server section. You must set the proxy path to iojs.

```
server {
    .. other configuration entries ...
    location /iojs/ {
    proxy_pass https://first.iovation.com/;
    }
    ... }
```

Save the configuration file and restart Nginx.

Example: Configuring a Reverse Proxy in Apache

Edit the Apache configuration file (**httpd.conf**) and enable the following modules:

- proxy_module
- proxy_http_module - to enable SSL forwarding

```
LoadModule proxy_module      modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
```

Add the following lines, in the main configuration or VirtualHost directive, assuming iojs is the URI.

```
SSLProxyEngine  On
ProxyPass /iojs/ https://first.iovation.com/
```

Save the configuration file and restart Apache.

Testing New Releases of the First-Party JavaScript

iovation periodically releases new versions of the first-party JavaScript. We strongly recommend testing these updates in our customer integration (CI) environment before we release production versions. We will notify you when we release updates, and provide up to two weeks to test. To test the updated JavaScript, complete the following steps:

1. Change the URLs in your proxy configurations to the following test URL in our CI environment: https://ci-first.iovation.com/
   When you have completed testing, restore the proxy configurations back to the production URL: https://first.iovation.com/
2. Test incoming transactions to verify the following:
   - There are no resource loading issues resulting in time-out responses back to the browser.
   - There are no changes in web page appearance, performance, or behavior.
   - There are no errors in the browser console.

Deploying First-Party JavaScript via Your Own Web Device Print Server

Installing a Web Device Print Server

IMPORTANT! If you will use iovation with more than one domain you must set up web device print servers for each domain.

1. Download the Multi-Domain Recognition deployment package from here:
   https://help.iovation.com/004_Download_SDKs/Download_Web_Device_Print_Components/Download_Web_Device_Print_Components
   This WAR file contains all of the files required to set up a web device print server in your domain.

2. If you don't already have a 64 bit Java web server **within the domain and port from which your pages are served**, establish one. You may also choose to set up a dedicated Java Server for the iovation JavaScript.

   a. If necessary, install a Java web server to a system that will be accessible to end users of your service, such as a server in a DMZ. You can use any Java web server. The server must reside under the same domain and port from which your pages are served.
   For example, you can download Apache Tomcat for your OS along with installation instructions here:
   http://tomcat.apache.org/download-70.cgi

   b. If you haven't already done so, configure Java to use 256 bit cryptography. You will need the Java Cryptography Extension (JCE). To get this from a browser, search for the current version of the Java Cryptography Extension (JCE) from your JDK vendor. Download the unlimited strength policy files. Once you have downloaded the version for your OS, copy the two Jar files to the following directory under your Java installation:

   ```
   $JAVA_HOME\jre\lib\security
   ```

   You must install the version of the JCE that corresponds to your Java version and vendor. For example, if you have installed Java 1.6, you must install version 6 of the JCE.

3. Deploy the web device print WAR file. For example, to deploy the WAR file to a Tomcat installation, place it under the /webapps folder under your Tomcat installation folder.

4. Create the file wdpoverride.properties and place in your CLASSPATH:

   a. Create a new file named wdpoverride.properties and place in your web server's CLASSPATH
   .

   b. In a text editor, add the text below to your **wdpoverride.properties** file. Replace `hostname.hostdomain.tld:port` with the correct server name and, if necessary, port number for your web device print server. You will also need to change wdp-first-service to the URI/Context on your server where the WAR file is installed:

```
parameter.content.server.path=base64_encode(wdp-first-service/resources/
static)
parameter.realip.server.host=base64_encode(mpsnare.iesnare.com)
# Origin of the JavaScript
parameter.wdp.server.host=base64_encode(hostname.hostdomain.tld:port)
# URL to a script that stores a token in your users' browser cache
parameter.ctoken.script.path=base64_encode(wdp-first-service/latest/logo.js)
parameter.wdp.js.versions=latest
parameter.wdp.js.version.latest=4.1.6
parameter.rtc.server.list=base64_encode(stun:stun.l.google.com:19302,
stun:stun3.l.google.com:19302,stun:stun2.l.google.com:19302,
stun:stun.stunprotocol.org:3478,stun:numb.viagenie.ca:3478,
stun:stun.vivox.com:3478,stun:stun.sip.us:3478,stun:stun.commpeak.com:3478,
stun:stun.barracuda.com:3478,stun:stun.epygi.com:3478)
```

5. Make the following changes to prevent memory leaks on your server from asynchronous logging appenders:

    a. In the WEB-INF/web.xml file for wdp-first-service, un-comment the `isLog4jAutoInitializationDisabled` parameter and set it to `true`:

    ```
    <context-param>
        <param-name>isLog4jAutoInitializationDisabled</param-name>
        <param-value>true</param-value>
    </context-param>
    ```

    b. Un-comment `Log4jServletContextListener` and `Log4jServletFilter`:

    ```
    <listener>
        <listener-class>org.apache.logging.log4j.web.
    Log4jServletContextListener</listener-class>
    </listener>

    <filter>
        <filter-name>log4jServletFilter</filter-name>
        <filter-class>org.apache.logging.log4j.web.Log4jServletFilter</filter-
    class>
    </filter>

    <filter-mapping>
        <filter-name>log4jServletFilter</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
        <dispatcher>ERROR</dispatcher>
        <dispatcher>ASYNC</dispatcher>
    </filter-mapping>
    ```

6. Start the web device print server. Monitor the logs on the Java web server to make sure that there are no errors.

7. To verify that the web device print server is running correctly, open the following URL in a browser:
   http://*yourserver*:*port*/wdp-first-service/latest/dyn_wdp.js
   For example, if your server is called **deviceprint.com** and your port number is **80**, open:

http://deviceprint.com:80/wdp-first-service/5.0.0/dyn_wdp.js
If the server is running correctly the JavaScript will display in the browser.

Configuring Device Recognition JavaScript

Overview

These topics walk you through configuring the iovation JavaScript. The iovation JavaScript collects device information that you can then submit to iovation as part of your ClearKey or Fraud Prevention implementation.

Configuring iovation JavaScript on Web Pages

To configure the iovation JavaScript, you must define settings for a configuration object that is used by the JavaScript. If the object is not defined, default values will be used.

The configuration object has various sections and looks something like the following:

```
/* Copyright(c) 2016, iovation, inc. All rights reserved. */
window.io_global_object_name = "IGLOO";
window.IGLOO = window.IGLOO || {
  "enable_flash" : false,
  "bbout_element_id" : "iobb",
  "bb_callback": null,
  "loader" : {
    "uri_hook" : "iojs",
    "version" : "general5",
    "subkey" : "5FExse+oA1134BhiwCF2EeQ1TfisPJGha4CpVG2nd7E="
  }
};
```

iovation will provide you with a configuration script in addition to the collection script. You can include these separately, combine them or in-line the definition on your page.

IMPORTANT!

It is critical that the *configuration* comes **before** the *iovation script* otherwise configuration variables will not be used once the script starts running.

Configuration script sections

iovation Global Object Name

Before delving into the options available, a quick word on the io_global_object_name, IGLOO, by default. In earlier versions of the scripts, iovation configured the script through the use of global variables. While this approach is simple enough, it can add a lot of additional variables to the window object which has the potential to collide with other scripts.

To remedy this, iovation has created an object that encapsulates all of the settings and functionality and you can change the ID to prevent potential collisions by changing the name.

```
window.io_global_object_name = "<your custom name>"
```

If you do change the value, references functions and values as follows:

```
window.<your custom name>.xxxxx
```

Change the following as well:

```
window.IGLOO = window.IGLOO ||
```

to match your new custom name.

iovation's configuration object has 2 main components:

- Generic settings that indicate how to retrieve a device print and restrictions on what information is collected, for instance Flash values
- Loader settings that indicate how to access first-party components and basic debug and versioning information

Each set of configuration options belongs in a specific section of the object:

```
/* Copyright(c) 2016, iovation, inc. All rights reserved. */
window.io_global_object_name = "IGLOO";
window.IGLOO = window.IGLOO || {
  // generic settings go here as
  // "option" : value
  ...
  "loader" : {
     // loader configuration options go here as
     // "option" : value
  }
};
```

Generic Configuration Options

Parameter	Type	Default Value	Description
`bbout_element_id`	string, optional		The ID of the HTML element to populate with the blackbox from the third-party JavaScript. If `bb_callback` is specified, this parameter has no effect.

`bb_callback`	function, optional		This JavaScript function is an event handler that is called when a collection method has finished updating a blackbox. This must be a function, not a string.
			
			Declare the function as follows:
			
			`"bb_callback" : function ( bb, complete) {`
			`   // code to process blackbox here`
			`}`
			
			The variables store the following:
			• `bb` – the updated value of the blackbox
			• `complete` – a boolean value indicating whether all the collection methods have completed.

`enable_rip`	boolean, optional	`true`	
`install_flash`	boolean, optional	`true`	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.
`min_flash_version`	decimal	10	Minimum version of Flash required for collection Flash values
`flash_needs_update_handler`	function, optional		If `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: `"flash_needs_update_handler" :`

`function () { alert( 'Install Flash!' ); }`

Loader Configuration Options

Parameter	Type	Default Value	Description

`uri_hook`	string, optional	`iojs`	Location of dynamic first party components. This should be a reference to the web directory being proxied. You can use relative or absolute references, but should not use a complete URL.
			
			For example, if your reverse proxy is accessed from `http://mysite.com/iov/wdp/....`, and your page is loaded from: `http://mysite.com/app/mypage.html`, you would use:
			
			`  "uri_hook" : "/iov/wdp"`
			
			If the reverse proxy is accessed at `http://mysite.com/app/iojs` and your page is at `http://mysite.com/app/page.html`, you might use:
			`  "uri_hook" : "iojs"`
			
			NOTE
			This should not be a URL (i.e. include http(s)://host:port) as this will fail and the scripts must be loaded from the same domain as the page.

`subkey`	string, optional		This will be an iovation assigned value that tracks requests from your site. This is primarily used for debugging and troubleshooting purposes.
`version`	string, required	`general5`	This is the version of the script to load. The value should either correspond to a specific version you wish to use, or one of the following aliases to get the latest version of the code: • `general5` - the latest stable version of the JavaScript

Parameter	Type	Default Value	Description
			• `early5` - the latest available version of the JavaScript `general5` and `early5` may be the same code, however, any new changes will be released on `early5` prior to `general5`. Once the new release has been vetted in production and deemed satisfactory, `general5` will be updated to match `early5`.

`trace_handler`	function, optional		This JavaScript function can be used to provide tracing messages for the script. This will provide information on the progress of the script and is useful for debugging purposes.
			
			Declare the function as follows:
			
			`"trace_handler" : function (message) {`
			`  // process/record trace message here`
			`}`
			
			where `message` is the trace message provided by the script.

`enable_legacy_compatibility`	boolean, optional	`false`	This flag is used to support using the older configuration variables and functions. This should only be used when migrating older integrations to use the newer script.

Example Configuration File

Putting together basic configuration settings, results in a file similar to:

```
/* Copyright(c) 2016, iovation, inc. All rights reserved. */
window.io_global_object_name = "IGLOO";
window.IGLOO = window.IGLOO || {
  "enable_flash" : false,
  "bbout_element_id" : "iobb",
  "loader" : {
    "uri_hook" : "/iojs",
    "version" : "general5",
    "subkey" : "5FExse+oA1134BhiwCF2EeQ1TfisPJGha4CpVG2nd7E="
  }
};
```

Collecting a Blackbox

Methods for Collecting a Blackbox

The iovation script creates a blackbox that you must collect and send to your server. This blackbox will contain all the device information collected from the various sources.

There are three ways to collect a blackbox.

- Populate a hidden form field using the `bbout_element_id` parameter.
- Define the `bb_callback` function to collect the blackbox as it is generated.
- Call the JavaScript function `window.IGLOO.getBlackbox` to obtain the blackbox. You can combine `getBlackbox` and either of the other two methods.

Implementing the Hidden Form Field Collection Method

You can define a hidden form field for the iovation script to populate. The blackbox will then be submitted along with other fields in the form. To use this method you must define `bbout_element_id` in the configuration script. 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 `bbout_element_id` parameter to the id of the hidden form field. For example, if the `id` attribute is set to `ioBlackBox`, set `bbout_element_id` to the following:

```
/* Copyright(c) 2016, iovation, inc. All rights reserved. */
window.io_global_object_name = "IGLOO";
window.IGLOO = window.IGLOO || {
  "enable_flash" : false,
  "bbout_element_id" : "ioBlackBox",
  "loader" : {
    "version" : "general5",
    "subkey" : "5FExse+oA1134BhiwCF2EeQ1TfisPJGha4CpVG2nd7E="
  }
};
```

Implementing the Callback Collection Method

The callback interface allows you to manage blackbox generation in a more event-driven manner. As blackbox collection progresses,the script fires update events as collection methods complete. 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.

In the JavaScript configuration parameters, set the `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, etc.) are complete

```
/* Copyright(c) 2016, iovation, inc. All rights reserved. */
window.io_global_object_name = "IGLOO";
window.IGLOO = window.IGLOO || {
  "enable_flash" : false,
  "bb_callback": function ( bb, complete ) {
                     var bb_field = document.getElementById( "bb" );
                     bb_field.value = bb;
                 },
  "loader" : {
    "version" : "general5",
    "subkey" : "5FExse+oA1134BhiwCF2EeQ1TfisPJGha4CpVG2nd7E="
  }
};
```

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

Implementing the getBlackbox Collection Method

The last method for obtaining a blackbox is to use the `getBlackbox` 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 `getBlackBox` collection method:

1. Call the function. For example:

   ```
   var blackbox_info = window.IGLOO.getBlackbox();
   ```

   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 loader configuration variable `trace_handler` parameter to troubleshoot. This will send output and status messages to your trace handler. A simple way to troubleshoot is to use something similar to:

```
/* Copyright(c) 2016, iovation, inc. All rights reserved. */
window.io_global_object_name = "IGLOO";
window.IGLOO = window.IGLOO || {
  "enable_flash" : false,
  bbout_element_id : "iobb",
  "loader" : {
    "version" : "general5",
    "subkey" : "5FExse+oA1134BhiwCF2EeQ1TfisPJGha4CpVG2nd7E=",
    "trace_handler": function ( msg ) {
                       console.log( msg );
                   }
  }
};
```

Blackbox Collection Examples

Overview

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

NOTE Besides the code samples below, it is important to set up the reverse proxy or web device print server as well. Make sure `uri_hook` in the `loader` configuration section is set appropriately. These examples assume the default configuration of `/iojs`.

Hidden Form Field Collection Example

Here is a sample web page that uses the hidden form field collection method.

```
<html>
<head>
  <title>Demo Integration Page</title>
  <meta charset="UTF-8">
</head>
<body>
  <form action="/do_ctd" method="post" name="loginForm" id="loginForm">
     <!-- Create a hidden field for the blackbox -->
     <input TYPE="hidden" NAME="ioBlackBox" id="ioBlackBox">
     <input type="submit" name="submit1">
  </form>
<!-- Include iovation JavaScript -->
<script language="javascript" src="config.js"></script>
<script language="javascript" src="iovation.js"></script>
</body>
</html>
```

```
<html>
<head>
  <title>Demo Integration Page</title>
  <meta charset="UTF-8">
</head>
<body>
  <form action="/do_ctd" method="post" name="loginForm" id="loginForm">
     <!-- Create a hidden field for the blackbox -->
     <input TYPE="hidden" NAME="ioBlackBox" id="ioBlackBox">
     <input type="submit" name="submit1">
  </form>
<!-- Include iovation JavaScript -->
<script language="javascript" src="config.js"></script>
<script language="javascript" src="iovation.js"></script>
</body>
</html>
```

Using the page layout from above, the hidden form field has an id of `ioBlackBox` that we set in `config.js` as `bbout_element_id:`

```
window.io_global_object_name = "IGLOO"
window.IGLOO = window.IGLOO || {
  "bbout_element_id" : "ioBlackBox",
  "loader" : {
    "version" : "general5",
    "subkey" : "5FExse+oA1134BhiwCF2EeQ1TfisPJGha4CpVG2nd7E="
  }
};
```

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>
<head>
  <title>Demo Integration Page</title>
  <meta charset="UTF-8">
</head>
<body>
<!-- Form 1 -->
  <form method=POST action="#">
    <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="io_bb"></input>
    <input type=submit name="Go second!"></input>
  </form>
<!-- Include iovation JavaScript -->
<script language="javascript" src="config.js"></script>
<script language="javascript" src="iovation.js"></script>
</body>
</html>
```

```
<html>
<head>
  <title>Demo Integration Page</title>
  <meta charset="UTF-8">
</head>
<body>
<!-- Form 1 -->
  <form method=POST action="#">
    <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="io_bb"></input>
    <input type=submit name="Go second!"></input>
  </form>

<!-- Include iovation JavaScript -->
<script language="javascript" src="config.js"></script>
<script language="javascript" src="iovation.js"></script>
</body>
</html>
```

Using the page layout from above, the callback function needs to populate both `io_bb` fields in the various forms. We do this using `bb_callback` in `config.js`:

```
window.io_global_object_name = "IGLOO"
window.IGLOO = window.IGLOO || {
  "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;
     },
  "loader" : {
    "version" : "general5",
    "subkey" : "5FExse+oA1134BhiwCF2EeQ1TfisPJGha4CpVG2nd7E="
  }
};
```

Get Blackbox Function Collection Example

This example illustrates how to use the blackbox method. Submission of the form initates the capture of the blackbox. You can alter the send function to post the blackbox via AJAX or perform some other operation. In the example, an alert displays the collected value.

```
<html>
<head>
  <title>Demo Integration Page</title>
  <meta charset="UTF-8">
</head>
<body>
  <form method=POST action="#">
    <input type=submit name="Go first!" onclick="return send_bb();"></input>
  </form>
  <script type="text/javascript">
     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 bb = "";
         try {
            bb = window.IGLOO.getBlackbox();
            alert( "bb: " + bb.blackbox );
         } catch (e) { alert( "Unable to get blackbox. " + e );
     }
   </script>
<!-- Include iovation JavaScript -->
<script language="javascript" src="config.js"></script>
<script language="javascript" src="iovation.js"></script>
</body>
</html>
```

As the functional interface is being used, there are no special configuration options required.

```
window.io_global_object_name = "IGLOO"
window.IGLOO = window.IGLOO || {
  "loader" : {
    "version" : "general5",
    "subkey" : "5FExse+oA1134BhiwCF2EeQ1TfisPJGha4CpVG2nd7E="
  }
};
```

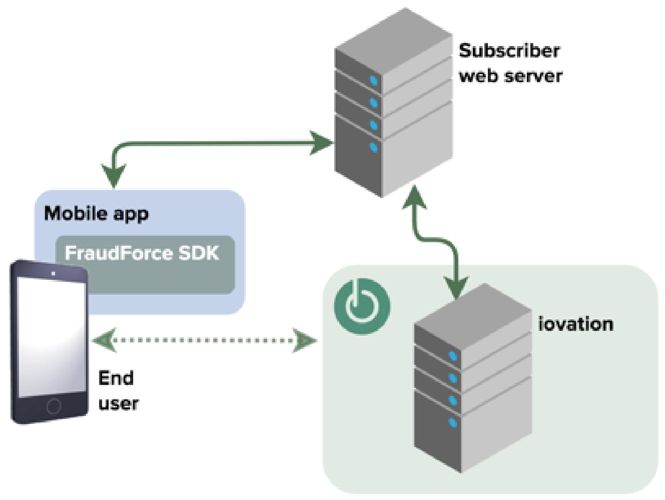
Integrating with Android Apps

Overview

Follow these steps to implement the iovation FraudForce SDK for Android.

About Mobile Integration

iovation identifies devices through information collected by an iovation FraudForce SDK run on an end-user's mobile device. The FraudForce SDK inspects the device to generate a blackbox that contains all available device information. This blackbox must then be transmitted to your servers to be used in a risk check.



The iovation SDK integrates with native and hybrid apps. Hybrid apps mix native code with content that runs inside a WebView.

Android Integration Files and Requirements

SDK Filename	fraudforce–lib–release–3.0.0.aar
Version	3.0.0
Package	com.iovation.mobile.android.FraudForce
Android SDK dependencies	Android SDK 4.1 or higher (SDK level 16)
Required Permissions	None
Optional Permissions	• BLUETOOTH • ACCESS_WIFI_STATE • READ_PHONE_STATE • ACCESS_FINE_LOCATION • GET_ACCOUNTS • ACCESS_NETWORK_STATE

NOTE If the permissions listed are not required by the application, the values collected using those permissions will be ignored. The permissions are not required to obtain a usable blackbox, but they do help obtain some unique device information.

Installing the SDK for Android

1. Download iovation-android-sdk-3.0.0.zip from GitHub: iovation Mobile SDK for Android.
2. Unzip iovation-android-sdk-3.0.0.zip.
3. Depending on your IDE, do one of the following:
   - **In Eclipse and Maven**, deploy the AAR file to your local Maven repository, using maven-deploy. For more information, see http://maven.apache.org/guides/mini/guide-3rd-party-jars-local.html
   - If you are using **Android Studio with Gradle**, add the *fraudforce–lib–release–3.0.0.aar* file to your application module's libs directory. Then, edit the *build.gradle* file in order to add the *libs* directory as a flat-file repository to the `buildscript` and `repository` sections. This makes the *fraudforce–lib–release–3.0.0.aar* file accessible to Gradle.

```
buildscript {
    repositories {
        flatDir {
            dirs 'libs'
        }
    }
}

repositories {
    flatDir {
        dirs 'libs'
    }
}
```

     Also in the application module's `build.gradle` file, make sure that *fraudforce-lib-release-3.0.0* is a compile-time dependency:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile(name:'deviceprint-lib-2.0.0', ext:'aar')
}
```

     Save the build.gradle file.

Integrating into Native Apps

To integrate into native apps:

1. In your Application class, import the `FraudForceManager` and `FraudForceConfiguration` objects.

```
import com.iovation.mobile.android.FraudForceConfiguration;
import com.iovation.mobile.android.FraudForceManager;
```

2. Create a configuration object with your subscriber key, and enable or disable network calls to iovation servers. Entering the subscriber key is strongly recommended for all integrations, and it is required for network connections.

   ```
   FraudForceConfiguration configuration = new FraudForceConfiguration.Builder()
       .subscriberKey([YOUR-SUBSCRIBER-KEY-HERE])
       .enableNetworkCalls(true) // Defaults to false if left out of configuration
       .build();
   ```

3. Initialize the `FraudForceManager` class using the generated `FraudForceConfiguration` object, and the context.

   ```
   FraudForceManager fraudForceManager = FraudForceManager.getInstance(context);
   fraudForceManager.initialize(configuration, context);
   ```

4. Call the `refresh()` method in the same activity or fragment where `getBlackbox()` will be called. The integrating application only needs to call this method on the Fragments where the `getBlackbox()` method will be called.

   ```
   FraudForceManager.getInstance().refresh(context);
   ```

   NOTE: This method calls updates the geolocation and network information, if enabled.

5. Do one of the following to build a blackbox:

   - To build a blackbox **asynchronously**, create an AsyncTask object to generate the blackbox off the main thread.

     ```
     private class FraudForceThread extends AsyncTask<Void, Void, String> {
         @Override
         protected String doInBackground(Void... voids) {
             return FraudForceManager.getInstance().getBlackbox(context);
         }

         @Override
         protected void onPostExecute(String blackbox) {
             // Integrator's code to store the blackbox
         }
     }
     ```

   - Then execute the FraudForceThread object to get the blackbox.

     ```
     new FraudForceThread().execute();
     ```

   - To build the blackbox **synchronously**, call the `getBlackbox(Context context)` function on a `FraudForceManager` object.

     ```
     String blackbox = FraudForceManager.getInstance().getBlackbox(context);
     ```

6. Call the `PrintThread` block and store the blackbox: `new PrintThread().execute();`

Integrating into Hybrid Apps

Hybrid App Workflow Overview

Integrate into hybrid apps by implementing the following workflow for collecting and sending blackboxes:

1. An HTML page loads in a WebView.
2. The user submits a transaction on the HTML page by submitting a form or completing another action.
3. This calls the `inject_bb` function, which creates a hidden iframe that calls the `iov://` URL. The iframe then deletes itself.
4. The `shouldOverrideUrlLoading` function inside of the WebView object in Java is called. This function detects the `iov://blackbox/fill#dom_id` URL. The `dom_id` is the ID of the object on the HTML page where the blackbox will be written, such as a hidden form field.
5. The `shouldOverrideUrlLoading` function runs JavaScript that automatically injects the blackbox into that object.
6. The blackbox is submitted to iovation.

Implementing Hybrid App Support

1. In your Application class, import the `FraudForceManager` and `FraudForceConfiguration` objects.

   ```
   import com.iovation.mobile.android.FraudForceConfiguration;
   import com.iovation.mobile.android.FraudForceManager;
   ```

2. Create a configuration object with your subscriber key, and enable or disable network calls to iovation servers. Entering the subscriber key is strongly recommended for all integrations, and it is required for network connections.

   ```
   FraudForceConfiguration configuration = new FraudForceConfiguration.Builder()
       .subscriberKey([YOUR-SUBSCRIBER-KEY-HERE])
       .enableNetworkCalls(true) // Defaults to false if left out of configuration
       .build();
   ```

3. Initialize the FraudForceManager class using the generated FraudForceConfiguration object, and the context.

   ```
   FraudForceManager fraudForceManager = FraudForceManager.getInstance();
   fraudForceManager.initialize(configuration, context);
   ```

4. In your WebView Activity's `onCreate()` function, set your WebView's `shouldOverrideUrlLoading()` function, as well as the `onPageStarted()` function.

```
wv.setWebViewClient(new WebViewClient() {
    @Override
    public void onPageStarted(WebView view, String url, Bitmap favicon) {
        FraudForceManager.getInstance().refresh(getContext());
        super.onPageStarted(view, url, favicon);
    }

    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        String[] ref = url.split("#");
        if (url.startsWith("iov://") && ref.length > 1 && ref[1] != null) {
        String injectedJavascript="javascript:(function() { " +
            "document.getElementById('" + ref[1] + "').value = '"
            + FraudForceManager.getInstance().getBlackbox(wv.getContext())
            + "';})()";
            wv.loadUrl(injectedJavascript);
            return true;
        }
        return false;
    }
});
```

5. On your HTML page, include a javascript function called `inject_bb` that injects an iframe with a call to the `iov://` URL.

```
function inject_bb(id) {
    var iframe = document.createElement('IFRAME');
    iframe.setAttribute('src', 'iov://blackbox/fill#' + id);
    iframe.name="ioOut";
    document.documentElement.appendChild(iframe);
    iframe.parentNode.removeChild(iframe);
    iframe = null;
}
```

6. Call the `inject_bb` function with the ID of the DOM object to inject the blackbox into for collection. The `shouldOverrideUrlLoading()` function will inject the blackbox. For example, set `ID` to a hidden form field where the blackbox will be stored. When the form containing the field is submitted, the blackbox is returned to your server back-end, and can then be sent to iovation to evaluate along with the transaction.

Network Calls

Starting with version 3.0.0, the SDK can make a network call to the iovation service. This enables additional functionality in the FraudForce SDK, including:

- Collecting additional network information
- Updating root detection configuration
- Collecting information on potentially high-risk applications on the device

By default this functionality is disabled and will need to be enabled in the configuration object. This feature requires a subscriber key, which your iovation Client Manager can provide.

Upgrading from the ioBegin Function

Prior to version 2.0.0, the iovation Mobile SDK for Android provided a blackbox via a call to `ioBegin`. This release deprecates `ioBegin`. This function is still available for backward compatibility. However, to take advantage of new features now and in the future, we strongly advise existing users to upgrade.

To upgrade:

1. Install the latest version of the SDK.
2. In your Activity's `onCreate()` function, add the `start` function to start background processing.
3. Add an `AsyncTask` code block with a sub-class called `PrintThread` that will asynchronously collect the blackbox using the `getBlackbox` function.

Compiling The Sample App in Android Studio

1. In Android Studio, select File | Open or click **Open Existing Android Studio Project** from the quick-start screen.
2. From the directory where you unzipped the SDK download, open the **android-studio-sample-app** directory.
3. In the project navigation view, open *src/main/java/com/iovation/mobile/android/sample/MainActivity.java.*
4. Right-click the file editing view and select *Run Main Activity*.

   IMPORTANT!

   If the option to run the module does not appear, select File | Project Structure and open the Modules panel. From there, set the Module SDK drop-down list to your target Android SDK version.

   Alternatively, you can right-click on the build.gradle file, and select **Run 'build'**
5. Select either an attached physical device, or an Android virtual device to run the app on. The app should now compile and launch.

When the app compiles successfully, you will see a view with a button that allows you to display a blackbox.

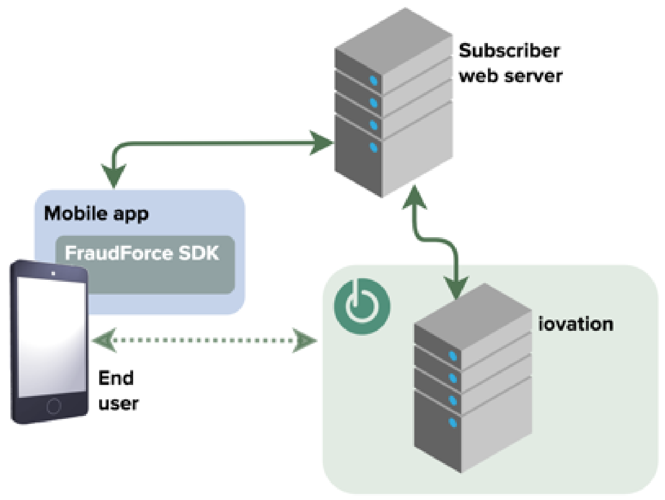
Integrating with iOS Apps

Overview

Follow these steps to implement the iovation Mobile SDK for iOS.

About Mobile Integration

iovation identifies devices through information collected by an iovation SDK run on an end-user's mobile device. The FraudForce SDK inspects the device to generate a blackbox that contains all device information available. This blackbox must then be transmitted to your servers to be used in a reputation check.



The FraudForce SDK integrates with native and hybrid apps. Hybrid apps mix native code with content that runs inside a web view.

Integration Files and Requirements

File	`FraudForce.framework`
Version	5.0.0
Required OS version	iOS 0.0 and higher
Supported Devices	iPhone 4S & up, iPod Touch 5th Gen & up, iPad 2 & up
Required Frameworks	CoreTelephony, Security, SystemConfiguration
Optional Frameworks	AdSupport, CoreLocation

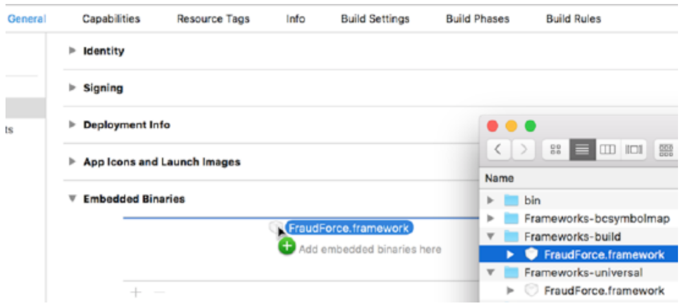
Installing the iovation FraudForce SDK for iOS

To install the iovation FraudForce SDK for iOS:

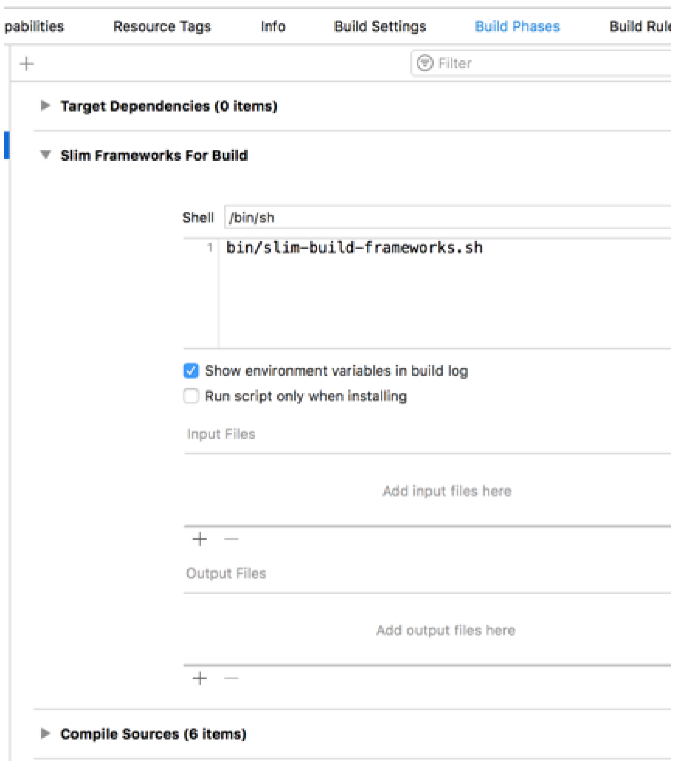
1. Download the SDK from GitHub and unzip it locally: https://github.com/iovation/deviceprint-SDK-iOS
2. Bring the universal framework into your project repository.

- Create a new directory named **Frameworks-universal**. We recommend this directory be located alongside the app's `.xcodeproj` file.
- Copy `FraudForce.framework` from the SDK distribution into your **Frameworks-universal** directory.

3. Prepare the staging area for build-specific frameworks.
   - Create a new directory, alongside the universal directory, named **Frameworks-build**.
   - Inside of **Frameworks-build**, create a new directory named **FraudForce.framework**.
     - **NOTE:** This is not intended to (initially) be a valid framework, rather just an empty directory that has the `.framework` extension.
4. In the Finder, drag `FraudForce.framework` from **Frameworks-build** into **Embedded Binaries** under the **General** pane in the Xcode target editor.



   - As a result, the framework is added to the *Linked Frameworks and Libraries* section.
   - Furthermore, `FraudForce.framework` appears in similarly named sections in the "Build Phases" pane.
5. Optionally add these frameworks if your app makes use of them (and if Auto Linking is off):
   - `AdSupport.framework` — If your app displays ads. Do not include if your app does not use the ad framework, because the App Store rejects apps that include the framework but don't use it.
   - `CoreLocation.framework` — If your app uses location monitoring. Do not include this framework unless your app requests geolocation permission from the user.
6. Add a pre-compile build phase to your application target.
   - Copy the shell script `slim-build-frameworks.sh` from the SDK distribution (`FraudForce SDK/build scripts`) into your project repository.
   - Select the + button in the **Build Phases** pane to create a *New Run Script Phase*.
     - The new phase is initially named *Run Script* and is positioned as the bottom-most (final) build phase.
     - The suggested name for this phase is *Slim Frameworks For Build*. Note that the name does not affect the function of this phase.
   - Reposition the new phase to precede the *Compile Sources* phase.
     - The phase will typically then be located below *Target Dependencies*.


pabilities
Resource Tags
Info
Build Settings
Build Phases
Build Rul
Filter
Target Dependencies (0 items)
Slim Frameworks For Build
Shell
Show environment variables in build log
Run script only when installing
Input Files
Add input files here
Output Files
Add output files here
Compile Sources (6 items)

```
/bin/sh
```

```
bin/slim-build-frameworks.sh
```

- Configure and confirm the new build phase.
  - The **Shell** text field should display the default value of **/bin/sh**.
  - In the text-input area, enter the project-relative path to the iovation-provided script `slim-build-frameworks.sh`.
- Ensure that the initial environment variables within this script are set to appropriate values based on the above steps.
  - Specifically, `INPUT_FRAMEWORKS_DIR` (Frameworks-universal) and `OUTPUT_FRAMEWORKS_DIR`(Frameworks-build) variables must be set to the absolute paths for the appropriate directories.
  - As provided, the script expects both directories to be located alongside the app's `.xcodeproj` file.

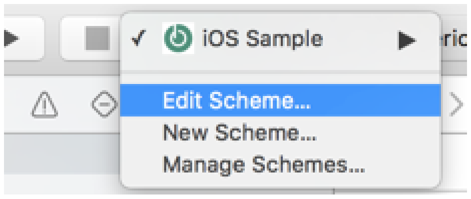
7. If your app has enabled Keychain sharing:

- Add *com.iovation.stm* to the list of Keychain Groups.
- Add the key `AppIdentifierPrefix` with the string value `$(AppIdentifierPrefix)` to your app's `Info.plist`.

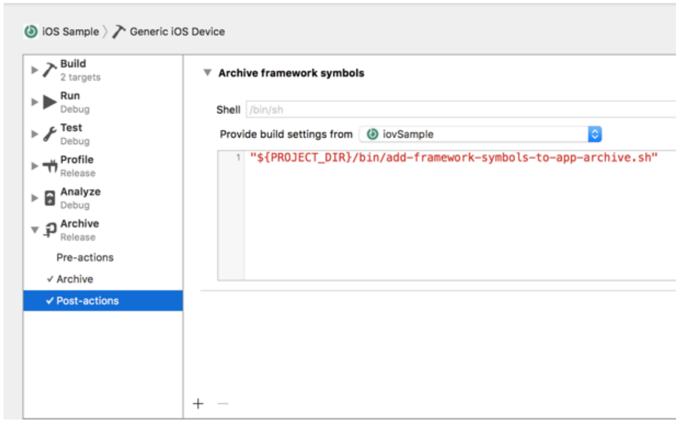
Submission Preparation

The FraudForce framework provides full support for Apple's bitcode technology. If your iOS app includes bitcode then additional configuration of your Xcode project is required to enable symbolication of FraudForce stack frames in the crash logs of your app. The necessary symbolic information must be included in the Xcode archive (i.e. `.xcarchive` bundle) of your application prior to its submission to the App Store.

1. Bring the framework `.bcsymbolmap` files into your project repository.
    - Create a new directory named **Frameworks-bcsymbolmap**. We recommend this directory be located alongside the app's `.xcodeproj` file.
    - Copy the `.bcsymbolmap` files from the SDK distribution (`FraudForce SDK/BCSymbolMaps`) into your **Frameworks-bcsymbolmap** directory.
        - There should be three such files (representing armv7, armv7s, and arm64 architectures).
2. Add a script to the *Archive* scheme of your application target.
    - Copy the shell script `add-framework-symbols-to-app-archive.sh` from the SDK distribution (`FraudForce SDK/build scripts`) into your project repository.
    - From the target popup button in the Xcode toolbar, ensure your application target is selected and chose **Edit Scheme...** from the popup menu.



    - In the scheme editor, expand **Archive** and select **Post-actions**.
        - Select the + button in order to create a new *Run Script Action*.
        - The new action is initially named *Run Script*. The suggested name for this action is *Archive Framework Symbols* (however the name does not affect the function of this action).



Archive framework symbols

Shell: /bin/sh

Provide build settings from: iovSample

```
"${PROJECT_DIR}/bin/add-framework-symbols-to-app-archive.sh"
```

    - Configure and confirm the new script action.
        - The default value for the Shell field is **/bin/sh**.
        - The **Provide build settings from** list should be set to the application target.
        - In the text-input area, enter the absolute path to the iovation-provided script `add-framework-symbols-to-app-archive.sh`.
            - For example, if the script is located in a `bin` directory that is alongside the app's `.xcodeproj` file then it should read, `"${PROJECT_DIR}/bin/add-framework-symbols-to-app-archive.sh"`
            - The quotes are critical as they ensure proper execution when path elements include spaces.
3. Ensure that the initial environment variables within this script are set to the values established in earlier steps.

   - The `BUILD_FRAMEWORKS_DIR` (Frameworks-build) and `INPUT_BCSYMBOLMAP_DIR` (Frameworks-bcsymbolmap) variables must be set to the absolute paths for the appropriate directories.
   - As provided, the script expects both directories to be located alongside the app's `.xcodeproj` file.
4. Optionally customize script output logging.
   - By default, Xcode does not handle or display output from scheme-based scripts, therefore the script redirects `stdout` and `stderr` to a (unique) log file in `/var/tmp`.
   - Change the output redirection or ignore all script output by removing redirection (which is declared near the start of the script).

Sample Projects

The download includes two sample Xcode projects that demonstrate the integration of the FraudForce SDK for iOS. These projects require at least Xcode 8 and iOS 9.0.

- The `iovSample/iovSampleSwift.Xcodeproj` project uses Swift to demonstrate two integration techniques: UIKit and WebKit. Each is implemented in its own view controller, and may be tested in a tabbed interface on a simulator or device.
- The `iovSample/iovSample.Xcodeproj` project uses Objective-C to demonstrate three integration techniques: UIKit, UIWebView, and WebKit. Each is implemented in its own view controller, and may be tested in a tabbed interface on a simulator or device.

Usage

Use the `FraudForce` API to enable the SDK to start collecting blackbox data asynchronously, and to generate a blackbox to submit to your back-end service.

1. Import it into your app delegate and call `+start` when the application becomes active:

   ```
   [FraudForce start];
   ```

   This method starts a low-priority thread that silently collects data from the device with minimal impact on your app.
2. Include your “Subscriber Key” in your application's `Info.plist`, using the key `IOVSubKey`.

   ```
   <key>IOVSubKey</key>
   <string>6S7EJX4BM7HuKUmriUOuQvXta9mBUs4tAVtGToP6tUY=</string>
   ```

   This is strongly recommended for all integrations, and it is required for network connections.
3. (optional) Provide a `FraudForceDelegate` object, which is required for network connections.

   ```
   - (BOOL)application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
   {
       [FraudForce delegation:self];
   ```

   ...

   ```
   - (BOOL)shouldEnableNetworkCalls
   {
   ```

```
    return YES;
}
```

4. To generate a new blackbox, call `+blackbox`:

   ```
   NSString *blackbox = [FraudForce blackbox];
   ```

5. Submit the blackbox to your service. The service should then send it to iovation to evaluate the transaction. See the iovSample Xcode projects included with the download for a sample implementation.

IMPORTANT! The blackbox returned from `+blackbox` should never be empty. An empty blackbox indicates that the protection offered by the system may have been compromised.

Asynchronous Processing Integration

Each of the native and hybrid integration methods described here benefit from starting the SDK asynchronous processing whenever your application becomes active. To do so, import `FraudForce` and call `+start` in the `-applicationDidBecomeActive:` method of your app delegate:

```
#import "SampleAppDelegate.h"
@import FraudForce;

-applicationDidBecomeActive:(UIApplication *)application
{
    [FraudForce start];
}
```

The `+start` method automatically registers a notification handler to be called when the application goes into the background. This handler asks the OS for time to finish any tasks it has running. To stop the SDK rather than have it finish tasks in the background, add a call to `+stop` in the `-applicationDidEnterBackground:` app delegate method.

Integrating Into Native Apps

To integrate into a native app using UIKit:

1. Start the asynchronous data collection as described.
2. Import `FraudForce` and call `+blackbox` wherever you need a blackbox:

   ```
   #import "SampleViewController.h"
   @import FraudForce;

   @implementation SampleViewController
   @property (strong, nonatomic) UILabel *blackbox;

   // Button press updates text field with blackbox value
   - (IBAction)changeMessage:(id)sender
   {
       self.blackbox.text = [FraudForce blackbox];
   }
   ```

```
@end
```

A more extensive example, including submitting a blackbox in an HTTP request, may be found in the `SampleUIKitViewController` class for both Swift (iovSampleSwift) and Objective-C (iovSample) in the sample Xcode projects included in the FraudForce SDK download.

Upgrading ioBegin Apps

Prior to version 4.0, the iovation Mobile SDK for iOS provided a blackbox via a call to `[iovation ioBegin]`. The 4.0 release deprecates `+ioBegin`. It is still available for backward compatibility, however to ensure going forward that you can take advantage of background processing and other new features, we strongly advise existing users to upgrade.

To upgrade:

1. Install the latest version of the SDK.
2. Start the asynchronous data collection.
3. Replace all calls to `+[iovation ioBegin]` with calls to `+[DevicePrint blackbox]`.

Integrating into Hybrid Apps

Integrating into UIKit Web View Hybrid Apps

If you use UIWebView to run web content inside your hybrid app, you can integrate iovation in two ways, both of which use `UIWebViewDelegate`. Choose which approach to follow based on your use case:

- **Scenario 1: Building and sending blackboxes when specific HTML pages load.** This is the simplest approach but only applies when you can link blackbox submission to the loading of a specific page.
- **Scenario 2: Building and sending blackboxes when users submit transactions.**

Scenario 1: Generating Blackboxes When Specific HTML Pages Load

Follow these steps to generate a blackbox every time a particular HTML page loads in the web view:

1. Start the asynchronous data collection.
2. On the HTML pages from which you will submit transactions, add a hidden form field that will store the blackbox.
3. In your `UIWebViewDelegate`'s `-webViewDidFinishLoad:` delegate method, call `-stringByEvaluatingJavaScriptFromString:` on the web view to inject the JavaScript that generates the blackbox and writes it to the hidden form field.

See the following complete example:

```
#import "SampleWebViewController.h"
#import FraudForce;

// Make it a UIWebViewDelegate.
@interface SampleWebViewController <UIWebViewDelegate>
@end

    @implementation SampleWebViewController

- (void)webViewDidFinishLoad:(UIWebView *)wv {
    NSString *bb = [FraudForce blackbox];
    [wv stringByEvaluatingJavaScriptFromString:[NSString
    stringWithFormat:@"document.getElementById('bbox').value = '%@'", bb
    ]];
}

@end
```

Scenario 2: Generating Blackboxes When Users Submit Transactions

If the HTML loaded into the UIWebView only sometimes needs a blackbox, such as when submitting a transaction, inject a JavaScript function into each page that that builds and submits the blackbox collected by the SDK. To do so:

1. Start the asynchronous data collection.
2. On the HTML pages from which you will submit transactions, add a hidden form field that will store the blackbox.
3. In your `UIWebViewDelegate`'s `-webViewDidFinishLoad:` delegate method, call `-stringByEvaluatingJavaScriptFromString:` on the web view to inject a JavaScript function into the page:

   ```
   [wv stringByEvaluatingJavaScriptFromString:@" \
       var Blackbox = { \
       injectInto: function(id) { \
       var iframe = document.createElement('IFRAME'); \
       iframe.setAttribute('src', 'iov://blackbox/fill#' + id); \
       document.documentElement.appendChild(iframe); \
       iframe.parentNode.removeChild(iframe); \
       iframe = null; \
       } \
       } \
   "];
   ```

   Calling this function from the HTML page triggers an out-of-band request with a URL where:

   - The scheme is `iov`
   - The URL fragment is the name of the hidden form field on the HTML page.
4. Next, add code to the `-webView:shouldStartLoadWithRequest:navigationType:` delegate method to capture requests to the custom `iov://` URL, generate the blackbox, and populate the hidden form field.

Add the following method to the `UIWebViewDelegate`:

```
- (BOOL)webView:(UIWebView *)wv
shouldStartLoadWithRequest:(NSURLRequest *)request
navigationType:(UIWebViewNavigationType)navigationType
{
    NSURL *url = request.URL;
    // Return true if it is not an iov:// URL.
    if (![url.scheme isEqualToString:@"iov"]) return YES;

    // Get the fragment identifying the hidden field to populate or return.
    NSString *frag = url.fragment;
    if (frag == nil) return YES;

    // Inject the blackbox into the hidden field.
    NSString *bb = [FraudForce blackbox];
    [wv stringByEvaluatingJavaScriptFromString:[NSString
        stringWithFormat:@"document.getElementById('%@').value = '%@'", frag, bb
    ]];

    // Return false to prevent a request and reload.
    return NO;
}
```

Consider adding code to check the domain and other attributes of the requesting document to ensure it is allowed to have a blackbox.

5. Then, anywhere on the page where you need a blackbox, such as on form submission, have the page call the injected function with the ID of the hidden field to populate with the blackbox:

```
<form name="txn" onsubmit="try { Blackbox.injectInto('bbox'); } catch(e) {} return
true;">
    <input type="hidden" id="bbox" name="bbox" />
    <!-- Other fields as required --->
    <input type="submit" />
</form>
```

NOTE The `try/catch` statement ignores errors if none of the `DevicePrint.injectInto` object hierarchy exists. This is essential if the HTML will be used outside an of your app.

Find a complete example in the `SampleUIWebViewController` class in the *iovSample* Xcode project included in the framework download.

Integrating into WebKit Apps

For a hybrid application using WebKit, create a JavaScript message handler and add it to the context of the web view.

1. Start the asynchronous data collection.

2. Build a view controller that implements the `WKScriptMessageHandler` protocol. Here's an example in Swift:

```
import UIKit
import WebKit
import FraudForce

class SampleWebKitViewController: UIViewController, WKScriptMessageHandler {

@IBOutlet var compatLabel: UILabel!
var webView: WKWebView?

override func viewDidLoad() {
    super.viewDidLoad()
    // Set up Blackbox.injectInto() via a user script.
    var js = "var Blackbox = { injectInto: function (id) {\n"
        + " window.webkit.messageHandlers.bb.postMessage(id)\n"
    + " } }\n"
    let userScript = WKUserScript(
    source: js,
    injectionTime: .AtDocumentEnd,
    forMainFrameOnly: true
)

// Set up bb notification.
let userContentController = WKUserContentController()
userContentController.addUserScript(userScript)
userContentController.addScriptMessageHandler(self, name: "bb")
let configuration = WKWebViewConfiguration()
configuration.userContentController = userContentController
// Create the web view.
        let webKitView = WKWebView(frame: self.view.bounds, configuration:
configuration)
        self.view.addSubview(webKitView)
        webKitView.translatesAutoresizingMaskIntoConstraints = true
        webKitView.autoresizingMask = [.flexibleHeight, .flexibleWidth]
        webView = webKitView
}
}
```

In this example, we added a script handler called `bb` and appointed the view controller itself as the delegate. The `Blackbox.injectInto()` function calls into this handler.

3. To hook up this script handler, add the following method to handle notifications from JavaScript running in the web view. This example assumes that an HTML ID for a hidden form field will be passed in the

message body:

```
func userContentController(
userContentController: WKUserContentController,
didReceiveScriptMessage message: WKScriptMessage
) {
    // Consider checking properties of message.webView.URL, such as the host
    // property, to ensure that it's a request from a known source.
    if message.name != "bb" { return }

    // Inject the blackbox.
    message.webView.evaluateJavaScript(
    "document.getElementById('\(message.body)').value = '\(Fraudforce.blackbox())'",
    completionHandler: nil
    );
    }
```

The last statement injects the blackbox from a call to `DevicePrint.blackbox()` into the hidden form field. Consider adding code to check the domain and other attributes of the requesting web page to ensure it is allowed to have a blackbox.

4. Then, invoke `FraudForce.injectinfo()` function any place in the HTML that you need a blackbox:

```
<form name="txn" onsubmit="try { DevicePrint.injectInto('bbox'); } catch(e) {}
return true;">
    <input type="hidden" id="bbox" name="bbox" />
    <!-- Other fields as required --->
    <input type="submit" />
</form>
```

NOTE The `try/catch` statement ignores errors if none of the `FraudForce.injectInto` object hierarchy exists. This is essential if the HTML will be used outside an of your app.

5. If you have full control over the HTML that will be loaded into WKWebView, including the name of a hidden field into which to inject a blackbox, append the code to inject the blackbox to the user script. Change the JavaScript variable declaration above to:

```
var js = "var Blackbox = { injectInto: function (id) {\n"
+ " window.webkit.messageHandlers.bb.postMessage(id)\n"
+ " } }\n"
+ "document.getElementById('bbox').value = '\(FraudForce.blackbox())'\n"
```

Find a complete example in the `SampleWKWebViewViewController` class for Swift (iovSampleSwift) and `SampleWebKitViewController` class for Objective-C (iovSample) in the sample Xcode projects included in the FraudForce SDK download.

Integrating with Mac OS Apps

Overview

Follow these steps to implement the iovation Mobile SDK for Mac OS X.

About Mac OS X Integration

iovation identifies devices through information collected by an iovation SDK run on an end-user's computer. The iovation SDK inspects the device to generate a blackbox that contains all available device information. This blackbox must then be transmitted to your servers to be used in a reputation check. The iovation SDK integrates with native and hybrid apps. Hybrid apps mix native code with content that runs inside a web view.

Integration Files and Requirements

File	`iovation.framework`
Version	4.2.0
Required OS version	64 bit OSX 10.7, Intel CPU
Optional Framework	CoreLocation

Installing the iovation SDK for Mac OS X

To install the iovation SDK for Mac OS X:

1. Log in to the iovation Intelligence Center, then download the SDK: https://help.iovation.com/004_Download_SDKs/Download_Desktop_SDKs/Download_SDK_for_Mac_OS_X . Unzip the SDK once it is downloaded.
2. In Xcode 6 or 7, select your app target, then display the **General** tab.
3. In the Finder, drag `iovation.framework` into the **Embedded Binaries** section in Xcode.
4. In the dialog that appears:
   - Select **Copy items if needed** to copy the framework file into your project's directory.
   - Select the targets in which you plan to use the framework.
   - Click **Finish**.
5. Select the **Build Settings** tab and set **Strip Debug Symbols During Copy** to **No**.
6. If your app has enabled the *Keychain Sharing* capability:
   - Add "com.iovation.stm" to its list of **Keychain Groups**.

  - Add the key `AppIdentifierPrefix` with the string value `$(AppIdentifierPrefix)` to your app's `Info.plist`.

7. Optionally add the `CoreLocation.framework` frameworks if your app uses location monitoring. Do not include this framework unless your application requests geolocation permission from the user.
8. If you encounter any missing symbol errors, you may need to display the General tab and add the following frameworks in the Linked Frameworks and Libraries section:
   - `libz.dylib`
   - `Cocoa.framework`
   - `IOBluetooth.framework`
   - `CoreWLAN.framework`
   - `Security.framework`
   - `IOKit.framework`
   - `DiskArbitration.framework`

Sample Project

SDK for Mac OS X download includes a sample Xcode project, iovSample/iovSample.Xcodeproj, that demonstrates the iovation SDK for OS X integration with a Cocoa Application as well as a WebKit WebView. This project requires Xcode 6 and Mac OS X 10.8.

Usage

Use the `DevicePrint` API to enable the iovation SDK to start collecting blackbox data asynchronously, and to generate a blackbox to submit to your back-end service.

1. Import the `DevicePrint` module into your app.
   - For Objective-C classes, import the module to files that need it:
     `@import iovation;`
   - For Swift apps, import the module in an Object-C bridging header.
2. Call `+start` when the application finishes launching. This method starts a low-priority thread that silently collects data from the device with minimal impact on your app.
   - In Object-C:
     `[DevicePrint start];`
   - In Swift:
     `Device.Print.start()`
3. To generate a new blackbox, call `+blackbox`.
   - In Objective-C:
     `NSString *blackbox = [DevicePrint blackbox];`
   - In Swift:
     `let blackbox = DevicePrint.blackbox()`

4. Submit the blackbox via HTTP to your service. The service should then send it to iovation to evaluate the transaction. See the *iovSample* Xcode project included with the download for a sample implementation.

IMPORTANT! The blackbox returned from `+blackbox` should never be empty. An empty blackbox indicates that the system may have been compromised.

Example

Import `DevicePrint` and call `+start` in the `-applicationDidFinishLaunching:` method of your app delegate:

```
#import "SampleAppDelegate.h"
@import iovation;
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    [DevicePrint start];
}
```

The SDK for OS X automatically registers listeners to stop the inspector on application termination. To stop the SDK rather than have it finish tasks in the background, add a call to `+stop` in the `-applicationWillTerminate:` method.

Integrating Into Native Apps

To integrate into a native app, import `DevicePrint` and call `+blackbox` wherever you need a blackbox:

```
#import "SampleViewController.h"
@import iovation;
@implementation SampleViewController
@property (strong, nonatomic) UILabel *blackbox;
// Button press updates text field with blackbox value
- (IBAction)changeMessage:(id)sender
{
    self.blackbox.text = [DevicePrint blackbox];
}
@end
```

A more extensive example, including submitting a blackbox in an HTTP request, may be found in the `SampleWindowController` class in the *iovSample* Xcode project included in the framework download.

Integrating into Hybrid Apps

For a hybrid application using WebKit, implement the WebFrameLoadDelegate and WebScripting informal protocols to set up a JavaScript object that passes method calls through to your Cocoa code. For example:

```
@import WebKit;
@import iovation;
@interface WebController : NSObject
@property (weak) IBOutlet WebView *webView;
@end
@implementation WebController
// Creates a "Blackbox" object in the JavaScript environment of the WebView.
- (void)webView:(WebView *)webView didClearWindowObject:(WebScriptObject *)windowObject
forFrame:(WebFrame *)frame
{
    [windowObject setValue:self forKey:@"Blackbox"];
}
// Maps the JavaScript injectInto() method to `injectBlackboxInto:`.
+ (NSString *)webScriptNameForSelector:(SEL)sel {
    if(sel == @selector(injectBlackboxInto:)) return @"injectInto";
    return nil;
}
// Allows JavaScript to call injectInto().
+ (BOOL)isSelectorExcludedFromWebScript:(SEL)sel {
    return sel != @selector(injectBlackboxInto:);
}
// Called from JavaScript, injects the blackbox into the specified HTML.
- (void)injectBlackboxInto:(NSString *)elemID {
    [self.webView.windowScriptObject evaluateWebScript:[NSString
        stringWithFormat:@"document.getElementById('%@').value = '%@'",
        elemID,
        [DevicePrint blackbox]
    ]];
}
@end
```

The last statement injects the blackbox from a call to `[DevicePrint blackbox]` into the hidden form field. Consider also adding code to `-webView:didClearWindowObject:forFrame:` to check the domain and other attributes of the requesting HTML page to ensure it is allowed to have a blackbox.

With these three methods in place, assign an instance of this class to the `frameLoadDelegate` property of a WebView. From then on, you can call `DevicePrint.injectInto()` to fill the hidden form field:

```
<form name="txn" onsubmit="try { DevicePrint.injectInto('bbox'); } catch(e) {} return true;">
    <input type="hidden" id="bbox" name="bbox" />
    <!-- Other fields as required --->
    <input type="submit" />
</form>
```

Note the use of try/catch to ignore errors if `DevicePrint.injectInto()` fails. This is essential if the HTML will be used outside an OS X WebKit web view.

You can find a complete example in the SampleWindowController class in the iovSample Xcode project included with the framework download.

Capability Details

To take advantage of the keychain sharing capabilities for your app and for this SDK, it’s important to verify App Services, Entitlements, Profiles, AppIDs, and Bundle IDs in the **Apple Developer Certificates, IDs, & Profiles** site:

- Verify that the *App ID* matches the *Bundle ID* and is valid.
- Verify that the *Team Provisioning Profile* has permissions for the App ID and reload it on testing machines. Any modifications to the App ID/Provisioning Profile requires a reload on the testing machines either via the **Download all** button in the **Accounts** settings in Xcode, or by manually downloading and installing the Provisioning Profile.
- Enable *App Services* for entitlements:
    1. Go to Capabilities | Keychain Sharing and click **ON**.
    2. Verify that all items under **Steps** have completed successfully.
    3. If there were no pre-existing entitlements, a new *appname.entitlements* file is created and added to the project.
    4. If the App will be distributed through the Mac App Store, enable **App Sandbox**.
    5. Verify that the Sandbox has network access and that the items under **Steps** have all completed successfully.
    6. Before running/archiving, clean the *Build* folder by holding the Option key and selecting Product | Clean build folder...
- Troubleshooting hints:
    - If the app does not open and logs errors regarding permissions or code signing issues, the keychain entitlement likely isn’t properly configured. A quick fix is to disable Keychain Sharing in the Capabilities, clean the build folder, rebuild the app, and run again.
    - Quitting Xcode and restarting can fix some signing issues, especially regarding Mac App and Developer ID.
    - Manually downloading the provisioning profile from the Apple Developer site is more reliable than Xcode trying to fix it for you.

Refer to the Apple developer documents "Adding Capabilities" and "Creating Your Team Provisioning Profile", as well as Technical Note 2415 for further information and troubleshooting tips.

Integrating with Windows Apps

Overview

Complete these steps to integrate iovation into native Windows apps.

About Windows Integration

To integrate iovation into native Windows applications, iovation provides a Windows DLL (dynamic link library). You must deliver this DLL along with your executable application.

The DLL does not send information directly to iovation. You must pass the blackbox to iovation from your back-end application server. How you send the blackbox to your server depends on your existing client/server interfaces.

Integration Files and Requirements

DLL file details	• **Filename:** StmOCX.dll • **Version:** ◦ 2.9.3 (Windows Vista and later, 64 bit only) ◦ 2.9.2 (Windows Vista and later, 32 bit only) ◦ 2.9.1 (Windows 98 - Windows XP) • **File size:** 2.9.3: 218 KB; 2.9.2: 198 KB
Class ID (CLSID)	`{7A0D1738-10EA-47FF-92BE-4E137B5BE1A4}`
Snare Custom Interface Identifier (IStm)	`{8380414D-5AA6-4B74-93EB-855D7361E6B4}`
Other Interfaces Implemented	• `IUnknown` • `IDispatch` • `IClassFactory` • `ISupportErrorInfo`
Library Dependencies	• rpcrt4.lib • **For 2.9.3:** Visual C++ Redistributable for Visual Studio 2015 from vc_redist.x64.exe (64bit) - see https://www.microsoft.com/en-us/download/details.aspx?id=48145 • **For 2.9.2**: Visual Studio 2013 C++ Redistributable Packages - vc120 (MSVCR120.DLL, MSVCP120.SLL) from vcredist_x86 (32bit) - see http://www.microsoft.com/en-us/download/details.aspx?id=40784

Downloading the iovation SDK for Windows

Download the iovation SDK for Windows here:

Download SDK for Windows

About the DLL Functions

You can call these functions directly from the DLL by loading the DLL and calling `GetProcAddress.`

Function	GetProcAddress value	Function Signature
`ioBegin`	`io_Begin`	`int (*ioBeginFn)( char ** )`
`ioVersion`	`io_Version`	`int (*ioVerFn)( char ** )`
`ioFreeData`	`io_FreeData`	`void (*ioFreeFn)( char **)`

DLL Integration Example

This example is in Visual C++. You can follow a similar methodology for Visual Basic, C, and other languages by loading the COM interface for the object and making the appropriate function calls.

```
#include <atlbase.h>
#include <iostream.h>
#include <windows.h>
 int useDll()
 {
       HMODULE hDLL = LoadLibrary( "StmOCX.dll" );

       if ( hDLL == NULL )   {
              cout << "Unable to load DevicePrint...." << endl;
              return 1;
       }
       try {
          ioFreeFn freeFn = (ioFreeFn) GetProcAddress( hDLL, "io_FreeData" );
          ioBeginFn beginFn = (ioBeginFn) GetProcAddress( hDLL, "io_Begin" );

          if ( beginFn ) {
              char *bb = NULL;
             if ( beginFn( &bb ) == 0 ) {
                  cout << "Blackbox: " << endl << bb << endl;
                 if ( freeFn )
                     freeFn( &bb );
                 else cout << "ioBegin failed" << endl;
              }
          } else cout << "Unable to locate io_Begin." << endl;
       }
       catch ( ... ) {
              cout << "Caught exception " << endl;
       }
       FreeLibrary( hDLL );
       return 0;
}
```

Using the ioBegin Function

The `ioBegin` function collects information about the device and generates an encrypted string containing this information.

Syntax

`int ioBegin( char** blackbox )`

Parameters

`Blackbox` - a string ranging in size from 1 KB to 4 KB.

Return Values

`0` on success. Any other value indicates an error.

Comments

- The blackbox that `ioBegin` returns should never be empty. An empty blackbox can indicate that system security has been compromised. The calling application should take corrective action, such as ending and restarting the application.
- When the buffer is no longer needed, call `ioFreeData` to release it.

Example

This example is in Visual C++. You can follow a similar methodology for Visual Basic, C, and other languages by loading the COM interface for the object and making the appropriate function calls.

```
#include <atlbase.h>
#include <iostream.h>
#include <windows.h>
 typedef int (*ioBeginFn)( char ** );
 typedef void (*ioFreeFn)( char **);
int main( int argc, char** argv)
{
    HMODULE hDLL = LoadLibrary( "StmOCX.dll" );
    ioBeginFn beginFn;
    if ( hDLL == NULL )
           return 1;
    try {
          freeFn = (ioFreeFn) GetProcAddress( hDLL, "io_FreeData" );
       beginFn = (ioBeginFn) GetProcAddress( hDLL, "io_Begin" );
      // this is the beginning of the call to get a blackbox
      if ( beginFn ) {
            char *bb = NULL;
        if ( beginFn( &bb ) == 0 ) {
            cout << "Blackbox: " << endl << bb << endl;
            // free the data when done
            if ( freeFn )
                  freeFn( &bb );
        }
    }  catch ( ... ) {  }
    FreeLibrary( hDLL );
    return 0;
}
```

Using the ioVersion Function

The `ioVersion` function returns the version of the iovation SDK. For example: `1.0.0`

`ioVersion` is informational and is used for testing and reporting purposes.

Syntax

`int ioVersion( char** version )`

Parameters

`version` – a pointer to a string containing the iovation SDK version, in the following format:

`<major version>.<minor version><revision>.<build>`

Return Values

`0` on success. Any other value indicates an error.

Comments

When the buffer is no longer needed, call `ioFreeData` to release it.

Example

This example is in Visual C++. You can follow a similar methodology for Visual Basic, C, and other languages by loading the COM interface for the object and making the appropriate function calls.

```
#include <atlbase.h>
#include <iostream.h>
#include <windows.h>
typedef int (*ioVerFn)( char ** );
typedef void (*ioFreeFn)( char **);
int main( int argc, char** argv)
{
    HMODULE hDLL = LoadLibrary( "StmOCX.dll" );
    ioBeginFn beginFn;
    if ( hDLL == NULL )
           return 1;
    try {
          freeFn = (ioFreeFn) GetProcAddress( hDLL, "io_FreeData" );
       verFn = (ioVerFn) GetProcAddress( hDLL, "io_Version" );
      // this is the beginning of the call to get a blackbox
      if ( verFn ) {
            char *version = NULL;
        if ( verFn( &version ) == 0 ) {
            cout << "Version: " << version << endl;
            // free the data when done
            if ( freeFn )
                  freeFn( &version );
        }
    }  catch ( ... ) {  }
    FreeLibrary( hDLL );
    return 0;
}
```

Using the ioFreeData Function

The `ioFreeData` function frees the memory buffer associated with `ioBegin` or `ioVersion`.

Syntax

```
void ioFreeData    ( char **          buf );
```

Parameters

`buf` – A pointer to the string returned by ioBegin or ioVersion.

Troubleshooting

Version 2.9.3

The following troubleshooting tips apply to version 2.9.3:

- Visual Studio 2015 has redesigned implementation of the C++ runtime libraries to be Universal. You must make sure that the 64-bit version is installed. For more information on this redesign, please see https://blogs.msdn.microsoft.com/vcblog/2015/03/03/introducing-the-universal-crt/
- Due to the requirements for the Universal CRT libraries, Windows must be completely patched and up-to-date.
- To run Debug versions of your app on other machines, Visual Studio 2015 must have both the Visual C++ and the Windows 10 SDKs installed.
- 32-bit Operating Systems will error on installation of 64-bit libraries, for example:
  0x81f40001 - Microsoft Visual C++ 2015 Redistributable (x64) - 14.0.23026 can only be installed on Windows XP SP1 (x64) and newer platforms.
- VS 2013 C++ libraries are not compatible with version 2.9.3 of the iovation Windows SDK and will return errors such as:
  The program can't start because MSVCP140.dll is missing from your computer. Try reinstalling the program to fix this problem

Preparing for Server-Side Integration

Overview

This topic helps you prepare for server-side integration.

Important Back-End Integration Considerations

When integrating iovation into your back-end servers, you must consider the following:

- Determine which REST or SOAP client you plan to use and, as needed, obtain sample code from iovation.
- Make sure you have defined and have access to the following important parameters:
  - The IP address for each end-user.
  - Unique identifiers for your end-users' accounts.
  - Your rule set names
- Make sure that you can update the following in your application when necessary to hit the appropriate iovation environments:
  - JavaScript URL
  - REST or SOAP URL
  - subscriberid
  - subscriberpasscode
- Make sure you have network connectivity to iovation's servers.
- Make sure to use DNS to resolve URLs.

Sample Code

iovation can provide you with sample code and WSDLs for all APIs. Contact your iovation Integration Engineer for samples.

SOAP Samples

If you are integrated via the legacy SOAP API, we provide samples for the following SOAP clients:

SOAP Client	Language
gSOAP	C, C++

SOAP Client	Language
AXIS and JAX-WS	Java
NuSOAP, PHP5	PHP
.NET	ASP .Net, C#
SOAP::Lite	Perl
Python Soap	Python
soap4r	Ruby

Obtaining IP Address from Each End-User

For best results with iovation, implement a method to obtain the IP address from each of your end-users. You will send this IP address to iovation using the `enduserip` parameter in the CheckTransactionDetails API.

The IP address is typically available as the server variable `REMOTE_ADDR` or in an HTTP header, such as `HTTP_CLIENT_IP` or `HTTP_X_FORWARDED_FOR`. Where the IP is stored depends on your infrastructure.

Load balancers may internally re-route traffic changing `REMOTE_ADDR` to the load balancer IP. In these cases, you can configure your load balancer to store the end user's IP in an HTTP header. Review your load balancer’s documentation and configuration to determine the HTTP header used to save the end user's IP.

Access to Production Servers

To ensure production access works correctly, set outbound access on port 443 to the following IPs:

- For web service calls:
  - 74.121.28.0/22
  - 103.21.244.0/22
  - 103.22.200.0/22
  - 103.31.4.0/22
  - 104.16.0.0/12
  - 108.162.192.0/18
  - 131.0.72.0/22
  - 141.101.64.0/18
  - 162.158.0.0/15

  - 172.64.0.0/13
  - 173.245.48.0/20
  - 188.114.96.0/20
  - 190.93.240.0/20
  - 197.234.240.0/22
  - 198.41.128.0/17

- If you are reverse proxying first-party JavaScript components:
  - 74.121.28.0/22
  - 66.151.32.32/27
  - 80.252.88.128/27

iovation and Cloudflare own these ranges and splits these IPs across data centers which is why the entire ranges need to be open.

Testing Connectivity

For SOAP integrations, you can test connectivity to our production servers by logging on to your production server and running the command:

```
telnet soap.iovation.com 443
```

If you are using proxy servers to connect to iovation, test the connection from your proxy servers, not your application servers.

iovation can also test connectivity to determine whether there is any network latency between the sites. To set this up, provide external IP addresses to iovation that we can periodically ping to monitor network responsiveness.

Using DNS to Route Requests

iovation uses a Global DNS service to manage traffic between its sites. This allows iovation to move traffic between data centers while we perform maintenance or troubleshoot production issues.

As a result, it is imperative that you do the following:

- Use DNS to route requests to iovation servers. **Do not use IP**.
- Make sure that your DNS servers obey DNS Time to live (TTL) records. iovation uses short DNS TTLs to more easily switch traffic between data centers. If your servers do not honor the DNS TTLs, your site could be affected by outages that impact one data center and not another.

NOTE Java servers tend to cache DNS entries to improve performance. This can prevent servers from being able to switch between our data centers during an outage. To prevent caching of DNS entries in your Java servers, set the following in your security file:

```
networkaddress.cache.ttl = 0
networkaddress.cache.negative.ttl = 0
```

Sending Requests With No Blackbox

We recommend that you send a request to iovation even when a blackbox is not captured. This enables you to use the Device not Provided rule, along with other anomaly rules, to add risk to transactions with no blackboxes. You can deny or review these transactions as needed (and change as appropriate).

Sending requests with no blackboxes also allows iovation to monitor collection rates and spot any issues. This allows iovation to determine whether there are any changes in collection rates. Changes might result from upgrades, or from fraudsters attempting to evade iovation detection.

Sending Transaction Risk Checks to iovation

REST Fraud Check API Overview

The REST Fraud Check API assesses risk for transactions based on the device and transaction details you provide. Use this RESTful API to:

- Send transactions to iovation, including device blackboxes, the name of the rule set to evaluate, an account ID, and the end-user's stated IP address
- Return risk assessment results
- Return device registration check results for iovation ClearKey
- Return detailed device and transaction information.

NOTE

This API provides equivalent functionality to our legacy CheckTransactionDetails SOAP API.

Sending Blackboxes to iovation

If you are implementing a web integration, you must combine the first-party and third-party blackboxes before sending them. The REST Fraud Check API only supports a single `blackbox` parameter. To append the blackboxes, combine them with a semicolon. You must combine them in the following order – first party blackbox, then third party blackbox:

```
1stpartyblackbox;3rdpartyblackbox
```

Resource URL

```
POST https://ci-api.iovation.com/fraud/v1/subs/{subscriberId}/checks
```

Where `{subscriberId}` is your iovation-assigned subscriber ID.

HTTP Headers

To successfully send a risk check to iovation, you must include the `Content-Type` and `Authorization` HTTP headers.

Content-Type

You must set the `Content-Type` header in each request to `application/json`.

```
Content-Type: application/json
```

This is currently the only supported format.

Authorization

iovation uses *Basic Access Authentication* to authenticate API requests. Format user names as follows:

`<subscriber ID>/<subscriber account>`

For example:

`1000/OLTP`

Pass the user name and password via the `Authorization` header. The password is your subscriber passcode. The header value should be formatted as follows:

`Basic (Base64 encoded string of <user name>:<password>)`

For example:

```
Authorization: Basic MzM0NzAyL09MVFC6UFbCVvU1MkY=
```

Request Body Parameters

The details of the request are sent in the body of the POST message. The request is formatted in JSON and contains the following parameters:

Parameter	Format	Description	Example
`accountCode`	String	Unique identifier for the end-user's account, or for the transaction.	`1221667954321`
`blackbox`	Blackbox String	The blackbox, which is an encoded string that includes all of the device information that iovation collects.	
`statedIp`	String	The stated IP address from the end-user's device.	For IP addresses, format as `nnn.n.n.nnn`, for example: `192.0.2.235`
`transactionInsight` ***FraudForce only***	Map <String, Object>	Transaction attributes that you can optionally send along with device information, that enable you to search for fraud based on identity data. **IMPORTANT!** Transaction Insight is available to FraudForce subscribers only.	`"billingCity": "Portland"`
`type` ***Required***	String	The identifier for the rule set to use for the transaction.	`login`

Transaction Insight Parameters

These attributes are optional. iovation stores them on your behalf, and you can then use them in certain business rules and reports. They are all available from the Transaction History report in the Intelligence Center, and from the Tracking Number Details view.

IMPORTANT!

Due to international regulations, some attributes are not accepted for subscribers based in the EU / UK. These are noted below with: **EU / UK? NO**

Attribute	Name	Description	API Format
`achRoutingNumber`	**ACH Routing Number**	ACH (Automated Clearing House) routing number for the customer's account. **EU / UK? NO**	9 digits, for example: `123456789`

Attribute	Name	Description	API Format
		Rules *Transaction Details Watch List*	
`alternateIp`	**Alternate IP**	Alternative representation of the customer's IP address. **EU / UK?** NO	A valid IPv4 or IPv6 address. IPv4 is preferred. For example, you can format an IPv4 address as follows: nnn.n.n.nnn, such as `192.0.2.235`. An example IPv6 address might look like the following: `2001:0db8:85a3:0000:0000:8a2e:0370:7334`
		Rules *Transaction Details Watch List*	
`billingCity`	**Billing City**	Name of the city from the customer's billing address.	Up to 80 bytes of UTF-8 characters, for example: `Portland`
		Rules • *Transaction Details Watch List* • *Transaction Details Comparison*	
`billingCountry`	**Billing Country**	2 character country code.	This must conform to ISO standard 3166, for example: `US`
		Rules • *Transaction Details Watch List* • *Transaction Details Comparison*	
`billingPostalCode`	**Billing Postal Code**	Postal code from the customer's billing address. **EU / UK?** NO	Up to 20 bytes of UTF-8 characters, for example: `97204`
		Rules • *Transaction Details Watch List* • *Transaction Details Comparison*	
`billingRegion`	**Billing Region**	Name of the region, for example a US state, from the customer's shipping address. This can be abbreviated or the full region name.	Up to 80 bytes of UTF-8 characters, For example: `OR`
		Rules	

Attribute	Name	Description	API Format
		• *Transaction Details Watch List* • *Transaction Details Comparison*	
`billingShippingMismatch`	**Billing / Shipping Mismatch**	Your determination whether customers' billing and shipping addresses match. iovation does not validate addresses but can track mismatches on your behalf.	Boolean: • `1` = true • `0` = false
		Rules *Billing / Shipping Mis-Match*	
`billingStreet`	**Billing Street**	Street address from the customer's billing address. **EU / UK? NO**	Up to 80 bytes of UTF-8 characters, for example: `111 SW 5th Ave`
		Rules *Transaction Details Watch List*	
`creditCardBin`	**Credit Card BIN**	The issuer identification number for the customer's credit card.	6 digits, for example: `471651`
		Rules • *Transaction Details Watch List*	
`email`	**Email Address**	Customer's email address. **EU / UK? NO**	Customer's email address.RFC 2822 email address format. For example: `account@domain.com`
		Rules • *Transaction Details Watch List* • *Transaction Details Comparison* • *Email Addresses per Account* • *Email Addresses per Device (Local)* • *Email Domain Watch List*	
`emailVerified`	**Email Address Verified**	Indicates that you have verified that an email associated with a customer account is valid.	Boolean: • `1` = true • `0` = false

Attribute	Name	Description	API Format
`eventId`	**Event ID**	ID associated with a transaction. This may be your own system tracking IDs. **Rules** *Transaction Details Watch List*	Up to 80 bytes of UTF-8 characters. For example: `168200gsd851`
`homePhoneNumber`	**Home Phone Number**	The customer's home phone number. **EU / UK?** NO **Rules** • *Transaction Details Watch List* • *Transaction Details Comparison* • *Phone Numbers Per Device* • *Devices Per Phone Number*	Plus sign (+) followed by 4-20 digits. This must be a valid phone number, including country code. For example, a US number must begin with 1, followed by the 10 digit number. You can include additional characters such as parentheses and hyphens. For example: `+15032246010`
`homePhoneSms Enabled`	**Home Phone SMS Enabled**	Indicates that you have verified that a home phone number associated with a customer is capable of receiving SMS text messages.	Boolean: • `1` = true • `0` = false
`homePhoneVerified`	**Home Phone Number Verified**	Indicates that you have verified that a home phone number associated with a customer account is valid.	Boolean: • `1` = true • `0` = false
`mobilePhoneNumber`	**Mobile Phone Number**	The user's mobile phone number. **EU / UK?** NO **Rules** • *Transaction Details Watch List* • *Transaction Details Comparison*	Plus sign (+) followed by 4-20 digits. This must be a valid phone number, including country code. For example, a US number must begin with 1, followed by the 10 digit number. You can include additional characters such as parentheses and hyphens. For example: `+15032246010`

Attribute	Name	Description	API Format
		• *Phone Numbers Per Device* • *Devices Per Phone Number*	
`mobilePhoneSmsEnabled`	**Mobile Phone SMS Enabled**	Indicates that you have verified that a mobile phone number associated with a customer is capable of receiving SMS text messages.	Boolean: • `1` = true • `0` = false
`mobilePhoneVerified`	**Mobile Phone Number Verified**	Indicates that you have verified that a mobile phone number associated with a customer account is valid.	Boolean: • `1` = true • `0` = false
`officePhoneNumber`	**Office Phone Number**	The customer's office phone number. **EU / UK?** NO	Plus sign (+) followed by 4-20 digits. This must be a valid phone number, including country code. For example, a US number must begin with 1, followed by the 10 digit number. You can include additional characters such as parentheses and hyphens. For example: `+15032246010`
		Rules • *Monetary Value per Account* • *Monetary Value per Device* • *Phone Numbers Per Device* • *Devices Per Phone Number*	
`officePhoneSmsEnabled`	**Office Phone SMS Enabled**	Indicates that you have verified that a work phone number associated with a customer is capable of receiving SMS text messages.	Boolean: • `1` = true • `0` = false
`officePhoneVerified`	**Office Phone Number Verified**	Indicates that you have verified that a work phone number associated with a customer account is valid.	Boolean: • `1` = true • `0` = false
`onlineId`	**Online ID**	The customer's online user name or alias. **EU / UK?** NO	Up to 80 bytes of UTF-8 characters.For example: `userID1234`
		Rules	

Attribute	Name	Description	API Format
		Transaction Details Watch List	
`referrerUrl`	**Referral URL**	Referral URL the customer followed to arrive at your website.	A valid RFC 3986 URL. For example:RFC 3986 URL. For example: `http://www.iovation.com`
		Rules *Transaction Details Watch List*	
`securityPin`	**Security PIN**	Personal identification number (PIN). **EU / UK?** NO	Up to 43 bytes of UTF-8 characters. You can send this as plain text or as a hashed value.
		Rules • *Transaction Details Watch List* • *Security PINs per Device*	
`securityPinType`	**Security PIN Type**	The type of security PIN. This may be useful if your PIN provider requires you to specify the type of PIN. If you use PINs, check with your PIN provider to determine whether you need to provide this, and if so, what the values might be. **EU / UK?** NO	Up to 40 bytes of UTF-8 characters.
		Rules *Transaction Details Watch List*	
`serviceId`	**Service ID(SiteID)**	The specific service or site from which the transaction originated. This is useful if you are managing rule sets for multiple websites or services.	Up to 255 bytes of UTF-8 characters.
		Rules *Transaction Details Watch List*	
`sessionAlias`	**Session Alias**	Stores an alias that represents the session identifiers for end users. Note that each session should be related to just one device.	Up to 64 UTF-8 characters. **This should not be the actual session ID**. We recommend sending this as a hashed or encrypted value.
		Rules • *Transaction Details Watch List* • *Transaction Details Comparison*	

Attribute	Name	Description	API Format
`shippingCity`	**Shipping City**	Name of the city from the customer's shipping address.	Up to 80 bytes of UTF-8 characters. For example: `Portland`
		Rules • *Transaction Details Watch List* • *Transaction Details Comparison*	
`shippingCountry`	**Shipping Country**	Name of the country from the customer's shipping address.	2 character country code. This must conform to ISO standard 3166. For example: `US`
		Rules • *Transaction Details Watch List* • *Transaction Details Comparison*	
`shippingPostalCode`	**Shipping Postal Code**	Postal code from the customer's shipping address. **EU / UK?** NO	Up to 80 bytes of UTF-8 characters. For example: `97204`
		Rules • *Transaction Details Watch List* • *Transaction Details Comparison*	
`shippingRegion`	**Shipping Region**	Name of the region, such as state, from the customer's shipping address.	Up to 80 bytes of UTF-8 characters. For example: `Oregon`
		Rules • *Transaction Details Watch List* • *Transaction Details Comparison*	
`shippingStreet`	**Shipping Street**	Street address from the customer's shipping location. **EU / UK?** NO	Up to 80 bytes of UTF-8 characters. For example: `111 SW 5th Ave`
		Rules • *Transaction Details Watch List* • *Transaction Details Comparison*	
`sku`	**SKU**	SKU for a product or service that the customer purchased.	Up to 80 bytes of UTF-8 character. For example:

Attribute	Name	Description	API Format
			`V4C3D5R2Z6`
		Rules *Transaction Details Watch List*	
`tenantID`	**Tenant ID**	Unique identifier for a specific business unit, location, affiliate, or other entity.	Up to 16 alphanumeric characters, case sensitive, may include `: ; / -` For example: `home-loans`
`upc`	**UPC**	UPC for a product or service that the customer purchased.	8, 12, or 13 digits, with any number of spaces and dashes. This must conform to the GTIN-8, UPC / GTIN-12, or GTIN-13 standard. For example: `01234567`
		Rules • *Transaction Details Watch List*	
`valueAmount`	**Monetary Amount**	Numeric value of a transaction.	Up to 15 digits, followed by a decimal, followed by two digits. For example: `10001.99`
		Rules • *Transaction Amount Range* • *Monetary Value per Account* • *Monetary Value per Device*	
`valueCurrency`	**Currency Type**	Currency type of the transaction.	3 character currency code. This must conform to ISO 4217. For example: `USD`
		Rules *Transaction Details Watch List*	

Example Request Body

```
{
  "statedIp": "192.0.2.235",
  "accountCode": "tv_15938",
  "blackbox": "0400UtAq9oNuGZINf94lis1zt",
  "type": "login",
  "transactionInsight" : {
      "email": "test@test.com",
      "homePhoneNumber" : "+12125551234"
  }
}
```

Response Properties

The fraud check response entity is a JSON object that contains multiple nested properties and entities.

IMPORTANT!

Please note the following:

- We do not provide the response properties in a specific order; the order that they are documented here may not be reflected in the response.
- We will add properties as needed over time, such as new device details. This may mean that responses include fields that are unknown to your applications; take care to integrate in such a way that unknown fields are ignored.

Response Structure

Response Parameters

The top level of the response includes the following parameters:

Attribute	Type	Description	Example
`accountCode`	String	The end user’s unique identifier, generally a username or ID. This value is echoed back from the request.	`1221667954321`
`details`	details entity	Structured JSON object that contains the device, IP and rules results.	
`id`	UUID	A unique ID for the request.	
`reason`	String, 50 characters	A user-specified value describing the rule that contributed the most to the result.	`Owned Evidence`
`result`	`[A|D|R]`	Result of the transaction risk or auth check, with a recommendation to allow (`A`), deny (`D`), or review (`R`) the transaction.	`D`
`statedIp`	String	The end-user's stated IP address.	`12.32.14.1`
`trackingNumber`	Long	A unique ID assigned to the transaction that can be used to locate the transaction in searches and reports.	`346060783728736616`

Response Entity Hierarchy

The response entities are included as follows:

NOTE

The following only lists the entity names. The ... indicates that the entity contains attributes that are not entities themselves.

```
{  ...
    details: {  ...
        device: {  ...
                blackBoxMetadata: { ... }
                browser: {  ...
                        flash: { ... }
                }
                mobile:   {  ...
                        build: { ... }
                        location: { ... }
                        system: { ... }
                        app: { ... }
                }
                registrationResult:   {  ...
                        matchStatus: { ... }
                        measureOfChange: { ... }
                }
        }
        verification: { ...
                 email:  [ { ... } ]
                 phone:  [ { ... } ]
                 }
        statedIp: {  ...
                ipLocation: { ... }
                botnet: { ... }
                }
        realIp:    {  ...
                ipLocation:        { ... }
                botnet: { ... }
                }
        ruleResults:  {  ...
                rules:   [ { ... } , ..., { ... } ]
                }
         machineLearning:  {  ... }
        }
}
```

details Entity

The `details` entity is an object that contains a variety of other objects representing elements such as device or transaction properties. Specific attributes are obtained by traversing the various entities to arrive at the value. All paths will start with the `details` entity as the first level.

Attribute	Type	Description
`device`	device entity	Device details captured by the iovation device recognition process.
`machineLearning`	machineLearning entity	SureScore results. This is only returned if you have subscribed to iovation SureScore.
`realIp`	ip entity	IP properties for the Real IP address.

Attribute	Type	Description
`ruleset`	ruleset entity	Ruleset and specific rule results, including scores.
`statedIp`	ip entity	IP properties for the end-user's stated IP address.
`verification`	verification entity	Email and Phone Verification properties.

device Entity

The `device` entity belongs to the `details` entity, and returns a number of device properties.

IMPORTANT!

* Business rules that are available only with FraudForce.

Attribute	Type	Description	Details / Example
`alias`	Long	iovation identifier for the device.	`712322039485795409`
		Rules • *Device List* • *Devices per Account* * • *Accounts Created per Device* * • *Accounts per Device* * • *Countries per Device* * • *Transactions per Device (Local)* * • *Global Transaction Device Velocity* * • *Account-Device Pair Transaction Count* * • *Monetary Value per Device* * • *Phone Numbers per Device* *	• *Devices per Phone Number* • *Email Addresses per Device (Local)* • *Security PINs per Device* • *Device New to Subscriber* • *New Device for Existing Account* • *Device Risk (Local)* • *Device Risk (Global)* • *Registered Account / Device Pair*
`blackboxMetaData`	blackboxMetadata entity	Entity that includes blackbox age properties.	
`browser`	browser entity *or* `null`	Entity that includes browser properties.	
`firstSeen`	Timestamp *ISO-8601 format:* `YYYY-MM-DDThh24:mi:ssZ` and expressed in GMT	Date/time the device was first seen by iovation.	`2017-06-01T14:43:21Z`
`isNew`	Boolean true if the device has never been seen by iovation, otherwise `false`.	Whether the device has ever been seen by iovation.	`false`
`mobile`	mobile entity *or* `null`	Contains device characteristics for mobile devices.	

`os`	String	Operating system of the device. **Rules** • *Transaction Details Watch List* • *Transaction Details Comparison* *	`WINDOWS NT 6.1`
`registrationResult`	registrationResult entity *or* `null`	For iovation ClearKey auth checks, contains device match results. This is only returned if you are a ClearKey subscriber and the ruleset includes the **Registered Account / Device Pair** rule.	
`screen`	String `###x###`	The screen resolution.	`1240X768`
`type`	String Options include: `MAC`, `PC`, `iOS`, `ANDROID`, `BLACKBERRY`, `CHROMEOS`, `HANDHELD_OTHER`, `IPHONE`, `IPOD`, `LINUX`, `MAC`, `NINTENDO`, `PLAYSTATION`, `UNIX`, `UNKNOWN`, `WINDOWS`, `WINDOWS_MOBILE`, `XBOX`, `N/A`	The hardware device make.	`MAC`

blackboxMetadata Entity

The `blackboxMetadata` entity belongs to the `device` entity and returns blackbox properties.

Attribute	Type	Description	Example / Details
`age`	Integer	Age of the blackbox, in seconds. **Rules:** *Blackbox Age*	`259200`
`timestamp`	ISO-8601 Date String, in GMT	Date / time the blackbox was created.	`2016-06-01T14:43:21Z`

browser Entity

The `browser` entity belongs to the `device` entity and includes various browser properties.

IMPORTANT!

* Business rules that are available only with FraudForce.

Attribute	Type	Description	Example / Details
`charset`	String	Accepted browser character sets from the HTTP header.	`ISO-8859-1,UTF-8;Q=0.7,*;Q=0.7`
`cookiesEnabled`	Boolean `true` if cookies are enabled and writable, `false` if cookies are not persistent or they are not writable	Whether JavaScript cookies are enabled.	`true`
`configuredLanguage`	String	Languages from the HTTP header that the browser will accept.	`EN-US,EN;Q=0.5`
`flash`	flash entity	Entity describing Flash properties.	
`language`	String	Browser default language. **Rules:** *Browser Language Watch List**	`EN-US`
`type`	String	Browser name. **Rules:** *Transaction Details Watch List*	`FIREFOX`
`timezone`	String, minutes from GMT	Browser timezone. **Rules:** *Timezone/Geolocation Mismatch*	`-480`
`version`	String	Browser version. **Rules:** *Transaction Details Watch List*	`3.6.3`

flash Entity

The `flash` entity belongs to the `browser` entity, and includes details on the installed version of Flash.

Attribute	Type	Description	Example / Details
`enabled`	Boolean `true` if Flash is enabled, otherwise `false`	Whether Flash is enabled.	`true`
`installed`	Boolean `true` if Flash is installed, otherwise `false`	Whether Flash is installed.	`true`
`storageEnabled`	Boolean `true` if Flash storage is enabled, otherwise `false`	Whether Flash storage is enabled.	`false`
`version`	String	Version of Flash that's installed.	`10.2.4.0`
		Rules: *Transaction Details Watch List*	

mobile Entity

The `mobile` entity belongs to the `device` entity and includes a number of mobile properties.

Attribute	Type	Description	iovation SDK Version	Example / Details
`app`	app identity	Includes properties that describe the app.		
`brand`	String	The brand of device, as determined by the manufacturer or carrier.	Android only, SDK 1.1+	`verizon`
`build`	build entity	Includes properties that describe the mobile build.		
`charging`	Boolean `true` if charging, otherwise `false`	Whether the device is currently plugged in and charging.	• Android SDK 1.1+ • iOS SDK 3.0+	`true`
`imei`	String	IMEI identifier.	Android only, SDK 1.1+	`358239059351439`
`location`	location entity	Describes location information received from the device		
`manufacturer`	String	Device manufacturer	Android only, SDK 1.1+	`samsung`
`model`	String	Device model name and model version. For Apple devices, this refers to the hardware identifier (such as iPhone6.1), not the public product model (such as iPhone 6s).	• Android SDK 1.0+ • iOS SDK 1.0, 3.0+	`iPhone6,1`
`orientation`	String Options: • `portrait` • `left` • `right`	***iOS only*** Physical orientation of the device (not the app) at the time of the transaction.	iOS SDK 3.0+	`portrait`
`screenResolution`	String	Device screen resolution, in pixels.	iOS SDK 3.0+	`1240X768`

`system`	System Entity	Describes system attributes of the mobile device

app Entity

The `app` entity belongs to the `mobile` entity, and for SDK integrations includes a number of parameters that describe the app with which the SDK is integrated.

IMPORTANT!

* Business rules that are available only with FraudForce.

Attribute	Type	Description	iovation SDK Version	Example / Details
`bundleId`	String	Application package name.		`com.iovation.droidid.demo`
`debug`	Boolean `true` if a debugger is attached, otherwise `false`	Whether a debugger is attached.	iOS only, SDK 3.0+	`false`
`exeName`	String	The filename of the app that processed the transaction.	• Android SDK 1.1+ • iOS SDK 3.0+	`/data/app/com.iovation.mobile.android.demo-1.apk`
		Rules: *Transaction Details Watch List* *Transaction Details Comparison**		
`marketId`	String	***Android only*** Unique identifier for the app.	Android only, SDK 1.1+	`BF95D92DAEB7B116`
`name`	String	The external application name.		`iovation Bank Demo`
`orientation`	String Options: • `portrait` • `right` • `left`	The physical orientation of the app (not the device) when the transaction was processed.	iOS only, SDK 3.0+	`portrait`
`procName`	String	The system process name.	iOS only, SDK 3.0+	`com.iovation.mobile.android.demo`
		Rules: *Transaction Details Watch List* *Transaction Details Comparison**		
`signerId`	String	The application signer ID.	Android only, SDK 1.1+	`1684797731`
`vendorId`	UUID	ID associated with a specific app producer; the vendor ID should	iOS only, SDK 3.0+	`77FC3F3E-321D-4187AAF6-17A0CDB4B9FC`

		be consistent across all apps by the same vendor.		
`version`	String	Software version of the application.	• Android SDK 1.1+ • iOS SDK 3.0+	`1.0.2`

build Entity

The `build` entity belongs to the `mobile` entity, and for Android devices only, includes parameters that describe the device.

Attribute	Type	Description	iovation SDK Version	Example / Details
`device`	String	Typically an internal project name, or may be an actual product name depending on the vendors naming standards.	Android only, SDK 1.1+	`new version`
`product`	String	The actual product name for the device.	Android only, SDK 1.1+	`Galaxy`

location Entity

The location entity belongs to the `mobile` entity, and includes properties that provide geolocation data on the device.

Attribute	Type	Description	iovation SDK Version	Example / Details
`altitude`	Float	Altitude reported by the device.	• Android SDK 1.1+ • iOS SDK 3.0+	`78.83`
`enabled`	Boolean `true` if location services are enabled, otherwise `false`	Whether location services are enabled.	iOS only, SDK 1.0+	`false`
`latitude`	Float	Latitude reported by the device.	• Android SDK 1.1+ • iOS SDK 3.0+	`45.519209`
`longitude`	Float	Longitude reported by the device.	• Android SDK 1.1+ • iOS SDK 3.0+	`-122.678744`
`timestamp`	ISO-8601 Date String	Geolocation timestamp.	• Android SDK 1.1+ • iOS SDK 3.0+	`2016-06-01T14:43:21Z`
`timezone`	String	Device timezone as determined by the OS.	• Android SDK 1.1+ • iOS SDK 3.0+	`America/Los_Angeles`
		Rules: *Timezone/Geolocation Mismatch* *Transaction Details Watch List*		

system Entity

The system entity belongs to the `mobile` entity, and provides details on the operating system and carrier.

IMPORTANT!

* Business rules that are available only with FraudForce.

Attribute	Type	Description	iovation SDK Version	Example / Details
`allowUnknownStore`	Boolean `true` if non-market apps are allowed, otherwise `false`	Whether apps can be installed from sources other than official app stores.	Android only, SDK 1.1+	`true`
`androidSDKLevel`	String	***Android only*** Version of the Google Android SDK that the device is using.	Android only, SDK 1.0+	`17`
`buildFingerprint`	String	Unique identifier for the system software build.	Android only, SDK 1.1+	`samsung/ jfltetmo/ jfltetmo:4.3/ JSS15J/ M919UVUEMK2`
`carrier`	String	The name of the mobile service provider.	• Android SDK 1.1+ • iOS SDK 3.0+	`VERIZON`
		Rules: *Transaction Details Watch List* *Transaction Details Comparison**		
`carrierCountryCode`	String	The country code associated with the mobile service provider.	• Android SDK 1.1+ • iOS SDK 3.0+	`US`
		Rules: *Mobile Carrier Country List** *Transaction Details Watch List*		
`cellularNetwork`	String	Type of cellular network.	iOS only, SDK 3.0+	`LTE`
`hostname`	String	The device's IP hostname.	Android only, SDK 1.1+	`android-ae64a827c9e6e266`

`jailrootDetected`	Boolean `true` if the device is jailbroken or rooted, otherwise `false`	Whether the device is jailbroken (iOS) or rooted (Android). **Rules:** *Jailbreak/Root Detected*	• Android SDK 1.2+ • iOS SDK 3.1+	
`localeCurrency`	String	The device currency locale.	• Android SDK 1.1+ • iOS SDK 3.0+	`USD`
`localeLang`	String	System locale and language setting.	• Android SDK 1.1+ • iOS SDK 3.0+	`en_US`
`networkOperatorCountry`	String	The country of the currently active operator, for example for a device that is traveling and connected to a different operator than usual.	Android only, SDK 1.1+	
`networkOperatorName`	String	The name of the currently active operator, for example for a device that is traveling and connected to a different operator than usual.	Android only, SDK 1.1+	
`osVersion`	String	Operating system version.	• Android SDK 1.0+ • iOS SDK 1.0+	`7.0.2`
`rootDetectionDisabled`	Boolean `true` if user attempted to hide rooting, otherwise `false`	User has attempted to hide detection of rooting or jailbreaking.	• Android SDK 1.1+ • iOS SDK 3.0+	`true`

`simulator`	Boolean `true` if the device is a simulator, otherwise `false`	Whether the transaction was initiated from a simulator.	• Android SDK 1.1+ • iOS SDK 2.0+	`true`
`uptime`	String	System uptime since last reboot or power up, in seconds.	• Android SDK 1.1+ • iOS SDK 3.0+	`94180.156`
`voipAllowed`	Boolean `true` if VOIP services are allowed, otherwise `false`	Whether the carrier allows voice over IP services for the device.	iOS only, SDK 3.0+	`true`

registrationResult Entity

The `registrationResult` entity belongs to the `device` entity and provides device match status and measure of change for iovation ClearKey auth checks.

IMPORTANT!

These are only returned for iovation ClearKey subscribers. The ***Registered Account / Device Pair*** business rule must be active in the rule set for the transaction in order to return ClearKey results.

Attribute	Type	Description	Example / Details
`matchStatus`	String [`MATCH` | `NO_MATCH` | `NONE_REGISTERED`]	Whether the device is registered to an account. **Rules:** *Registered Account / Device Pair*	`NONE_REGISTERED`
`measureOfChange`	String [`NONE` | `LOW` | MEDIUM]	When the `matchStatus` attribute returns a `MATCH` result, `measureOfChange` attribute returns an assessment of the degree of difference between the device from the new transaction and the closest matching device that is already paired with the account. **Rules:** *Registered Account / Device Pair*	`MEDIUM`

ip Entity

The `ipLocation` entity can belong to either the `statedIp` or `realIp` entity, and includes IP and ISP details.

IMPORTANT!

* Business rules that are available only with FraudForce.

Attribute	Type	Description	Example / Details
`address`	String	The IP Address.	`12.32.14.1`
		Rules: • *IP Address Mismatch* • *IP Address Distance** • *Transactions per IP** • *Transaction Details Comparison** • *IP Address Range**	• *IP Address List* • *ISP Organization Watchlist* • *Tor Exit Node IP List* • *Timezone/ Geolocation Mismatch* • *IP Address* Risk
`botnet`	botnet entity	Returns botnet score, intensity, and lastseen date.	
`isp`	String	Internet service provider of the stated IP address. Your account must be configured to return this information. The default is to not return this.	`AT&T WORLDNET SERVICES`
		Rules: • *ISP Mismatch* • *ISP List*	
`loc`	loc entity	Returns location details for the IP address. Your account must be configured to return this information. The default is to not return this.	
`org`	String	ISP Organization that the stated IP address is assigned to. Your account must be configured to return this information. The default is to not return this.	`AT&T WORLDNET SERVICES`
		Rules: *ISP Organization Watchlist**	
`proxy`	[`SATELLITE` | `PROXY`] *or* `null`	Indicator or special attributes for the IP address. This can be satellite or proxy. Your account must be configured to return this information. The default is to not return this.	`PROXY`
		Rules: *Proxy in Use*	

`source`	[`IOVATION` | `SUBSCRIBER`] or *`null`*	Whether the IP address is the stated address from the end-user, or discovered by iovation.	`SUBSCRIBER`

botnet Entity

The `botnet` entity belongs to either the `statedip` or `realip` entity, and includes botnet details for the IP address.

Attribute	Type	Description	Example / Details
`score`	Integer	The botnet risk score for the transaction. This is a range between 1 and 10 that you can factor into your transaction scores.	`9`
		Rules: *Minimum Botnet Score*	
`intensity`	Integer	A measurement of the frequency of suspected botnet activity associated with the IP address. This is a ranking between 0-10 where 0 indicates minimal botnet activity and 10 is a very high frequency of botnet activity.	`2`
`lastseen`	String	When botnet activity was last seen for the IP address, formatted as follows based on ISO 8601 and where `000Z` denotes that the time is in the 0 UTC offset timezone: `YYYY-MM-DDTHH:MM:SS.000Z`	`2019-01-28T21:43:11.000Z`

ipLocation Entity

The `ipLocation` entity belongs to an `ip` entity, and includes geolocation details for the IP address.

IMPORTANT!

* Business rules that are available only with FraudForce.

Attribute	Type	Description	Example / Details
`city`	String	City associated with the IP address.	`PORTLAND`
		Rules: *Transaction Details Watch List* *Transaction Details Comparison* *	
`country`	String	Country associated with the IP address.	`UNITED STATES`
		Rules: • *Country List* • *Language and Country Risk (Local)* * • *Geolocation Mismatch*	
`countryCode`	String	Country code associated with the IP address.	`US`
		Rules: • *Country List* • *Language and Country Risk (Local)* * • *Transaction Details Comparison* *	
`latitude`	Float	Lattitude associated with the IP address.	`-10.45`
`longitude`	Float	Longitude associated with the IP address.	`4.32`
`region`	String	State/region name associated with the IP address. To ensure that regions work as expected in business rules, **do not abbreviate them**.	`GEORGIA`
		Rules: • *Geolocation Mismatch* • *Transaction Details Watch List* • *Transaction Details Comparison* *	

ruleResults Entity

The `ruleResults` entity belongs to the `details` entity and returns high-level ruleset results.

Attribute	Type	Description	Example / Details
`score`	Integer	Total transaction score for the ruleset.	`-200`
`rulesMatched`	Integer	Number of rules that were triggered.	`3`
`rules`	Array of rules entity	Separate entity with details on each rule that was triggered.	

rules Entity

The `rules` entity belongs to the `ruleset` entity and provides the details for a rule that fired in the rule set.

Attribute	Type	Description	Example / Details
`type`	String	Type of rule that triggered.	`Accounts Per Device`
`reason`	String	The reason associated with the rule that matched. This value is supplied by you.	`trusted evidence`
`score`	Integer	Score contribution from the rule.	`-100`

verification Entity

For Email and Phone Verification subscribers, the `verification` entity includes arrays of `email` and `phone` entities containing the email and phone verification results.

Attribute	Type	Description
`email`	array of email entity	Contains email verification results.
`phone`	array of phone entity	Contains phone number risk scores.

email Entity

The `email` entity contains email verification results.

Attribute	Type	Description	Example
`address`	String	The email address that was submitted for verification.	`mailbox@domain.com`
`age`	String	A rough scale of how old the email address is. Values will be `Low`, `Medium`, `High`.	`Low`
`containsDomainIssue`	Boolean	The domain portion of the email address (the string after the @ symbol) does not exist or cannot receive email. • `true` : There is an issue with the domain. • `false` : There is no issue with the domain.	`false`
`containsMailboxIssue`	Boolean	The mailbox portion of the email address (the string before the @ symbol) does not exist or cannot receive mail. • `true` : The is an issue with the mailbox. • `false` : There is no issue with the mailbox.	`false`
`containsSyntaxIssue`	Boolean	The email address, as submitted, contains syntactic errors such as typos. • `true` : There is an issue with the syntax. • `false` : There is no issue with the syntax.	`false`
`dateFirstSeen`	String	Formatted as follows based on ISO 8601 and where `000Z` denotes that the time is in the 0 UTC offset timezone: `YYYY-MM-DDTHH:MM:SS.000Z`	`2019-01-28T21:43:11.000Z`
`daysSinceFirstSeen`	String	Number of days since the email was first seen on the web.	`32`
`disposable`	Boolean	The email address is temporary. • `true` : The email address is disposable. • `false` : The email address is not disposable.	`false`
`domainNotAllowed`	Boolean	The domain is either spam or is malicious. • `true` : • `false` :	`true`
`popularity`	String	An assessment of the number of sites at which the email has been seen. This may be `high`, `medium`, or `low.`	`medium`
`usernameNotAllowed`	Boolean	The username is either spam or is malicious. • `true` : • `false` :	`true`

`velocity`	String	An assessment of the frequency with which the email is seen at websites. This may be `High`, `Medium`, or `Low.`	`Medium`

phone Entity

The `phone` entity contains phone number verification results.

Attribute	Type	Value	Description
`number`	String	The phone number that was verified..	`+15554443333`
`phoneRisk`	String	An assessment of risk for the phone number. This is an integer between 1 and 100	`70`
`type`	array of Strings	Contains the field types from the request the number was sent as. This can be: `home`, `mobile`, or `office`	`office`

machineLearning Entity

For SureScore subscribers, the `machineLearning` entity returns the `mlvalue1` entity, which contains the SureScore outcome for the transaction.

Attribute	Type	Description
`mlvalue1`	mlvalue1 entity	Contains the value property, which returns a SureScore outcome for the transaction.

mlvalue1 Entity

The value entity contained by the `mlvalue1` entity returns a SureScore outcome for the transaction.

IMPORTANT!

This is available to SureScore subscribers only.

Attribute	Type	Description	Example / Details
`value`	Integer between `-10000` and `10000`	A score ranging from -10000 to 10000. -10000 indicates iovation's prediction that the transaction will be fraudulent while 10000 indicates iovation's prediction that the transaction will be trustworthy.	`9000`
		Rules: *SureScore Range*	

Example Response

Example Webprint Response

```
{
  "id": "06511936-8623-3cd4-b70f-6b852f7582f2",
  "result": "D",
  "reason": "Evidence found",
  "statedIp": "1.1.1.1",
  "accountCode": "test",
  "trackingNumber": 2000040009070100,
  "details": {
    "device": {
      "alias": 10003920005100080,
      "blackboxMetadata": {
        "age": 16,
        "timestamp": "2018-04-24T05:02:08Z"
      },
      "browser": {
        "cookiesEnabled": true,
        "configuredLanguage": "EN-US,EN;Q=0.9",
        "flash": {
          "installed": true,
          "version": "29.0.0"
        },
        "language": "EN-US",
        "type": "CHROME",
        "timezone": "480",
        "version": "65.0.3325.181"
      },
      "firstSeen": "2017-02-16T20:15:17.801Z",
      "isNew": false,
      "os": "WINDOWS NT 10.0",
      "registrationResult": {
        "matchStatus": "NONE_REGISTERED"
      },
      "screen": "1080X1920",
      "type": "WINDOWS"
    },
    "statedIp": {
      "address": "1.1.1.1",
      "isp": "CLOUDFLARE INC",
      "ipLocation": {
        "city": "BRISBANE",
        "country": "AUSTRALIA",
        "countryCode": "AU",
        "latitude": -27.46758,
        "longitude": 153.02789,
        "region": "QUEENSLAND",
        "botnet": {
           "intensity": 1,
           "score": 9,
           "lastSeen": "2019-04-09T06:19:18.000Z"
         }
      },
      "parentOrganization": "APNIC AND CLOUDFLARE DNS RESOLVER PROJECT",
      "source": "subscriber"
    },
   "realIp": {
      "address": "74.121.28.132",
      "isp": "IOVATION  INC.",
      "ipLocation": {
        "city": "PORTLAND",
        "country": "UNITED STATES",
        "countryCode": "US",
        "latitude": 45.51815,
        "longitude": -122.67416,
        "region": "OREGON",
        "botnet": {
           "intensity": 1,
           "score": 9,
           "lastSeen": "2019-04-09T06:19:18.000Z"
         }
      },
      "parentOrganization": "IOVATION  INC.",
      "source": "iovation"
    },
    "verification": {
      "email": [
        {
          "address": "bob@bob.com",
          "containsDomainIssue": false,
          "containsMailboxIssue": false,
          "containsSyntaxIssue": false,
          "age": "High",
          "dateFirstSeen": "2019-02-05T14:54:09.634Z",
          "daysSinceDateFirstSeen": 0,
          "velocity": "Medium",
          "popularity": "Low",
          "disposable": false,
          "domainNotAllowed": false,
          "userNameNotAllowed": false
        }
      ],
      "phone": [
        {
          "number": "+15035551212",
          "riskScore": 100,
          "type": [
            "home",
            "work"
          ]
        },
        {
          "number": "+13055551212",
          "riskScore": 100,
          "type": [
            "mobile"
          ]
        }
      ]
    },
    "ruleResults": {
      "score": -150,
      "rulesMatched": 2,
      "rules": [
        {
          "type": "Evidence Exists",
          "reason": "Evidence found",
          "score": -100
        },
        {
```

Example Mobile SDK Response

```
{
  "id": "5f71f112-20b0-325e-bf6d-1fbb2f689305",
  "result": "D",
  "reason": "Evidence found",
  "statedIp": "1.1.1.1",
  "accountCode": "test",
  "trackingNumber": 100063000005961100,
  "details": {
    "device": {
      "alias": 112348694602633100,
      "blackboxMetadata": {
        "age": 49,
        "timestamp": "2018-04-24T05:05:05Z"
      },
      "firstSeen": "2018-04-24T17:05:54.797Z",
      "isNew": true,
      "mobile": {
        "charging": false,
        "model": "IPHONE7,2",
        "orientation": "PORTRAIT",
        "screenResolution": "750X1334",
        "app": {
          "bundleId": "COM.MYAPP.MOBILE",
          "exeName": "MYAPP",
          "debug": false,
          "name": "MYAPP",
          "orientation": "PORTRAIT",
          "procName": "MYAPP",
          "vendorId": "01234D5-F067-8A90-1234-56F78D90A321",
          "version": "9388"
        },
        "location": {
          "altitude": 232.8735618591309,
          "enabled": true,
          "latitude": 40.14798992317183,
          "longitude": -88.25725137264305,
          "timezone": "AMERICA/CHICAGO"
        },
        "system": {
          "carrier": "VERIZON",
          "carrierCountryCode": "US",
          "cellularNetwork": "LTE",
          "jailRootDetected": false,
          "localeCurrency": "USD",
          "localeLang": "EN_US",
          "osVersion": "10.2.1",
          "simulator": false,
          "uptime": 189878.43,
          "voipAllowed": true
        }
      },
      "os": "IPHONE 10.2.1",
      "registrationResult": {
        "matchStatus": "NONE_REGISTERED"
      },
      "type": "IPHONE"
    },
    "statedIp": {
      "address": "1.1.1.1",
      "isp": "CLOUDFLARE INC",
      "ipLocation": {
        "city": "BRISBANE",
        "country": "AUSTRALIA",
        "countryCode": "AU",
        "latitude": -27.46758,
        "longitude": 153.02789,
        "region": "QUEENSLAND",
        "botnet": {
            "intensity": 1,
            "score": 9,
            "lastSeen": "2019-04-09T06:19:18.000Z"
          }
      },
      "parentOrganization": "APNIC AND CLOUDFLARE DNS RESOLVER PROJECT",
      "source": "subscriber"
    },
    "realIp": {
      "address": "1.1.1.1",
      "isp": "CLOUDFLARE INC",
      "ipLocation": {
        "city": "BRISBANE",
        "country": "AUSTRALIA",
        "countryCode": "AU",
        "latitude": -27.46758,
        "longitude": 153.02789,
        "region": "QUEENSLAND",
        "botnet": {
            "intensity": 1,
            "score": 9,
            "lastSeen": "2019-04-09T06:19:18.000Z"
          }
      },
      "parentOrganization": "APNIC AND CLOUDFLARE DNS RESOLVER PROJECT",
      "source": "subscriber"
    },
    "ruleResults": {
      "score": -150,
      "rulesMatched": 2,
      "rules": [
        {
          "type": "Evidence Exists",
          "reason": "Evidence found",
          "score": -100
        },
        {
          "type": "Evidence Exists",
          "reason": "Any subscriber evidence",
          "score": -50
        }
      ]
    },
    "verification": {
      "email": [
        {
          "address": "test@test.com",
          "containsDomainIssue": false,
```

HTTP Response Codes

The risk check API will return success or failure information through the HTTP status codes returned. For success messages, the response body will contain the response elements. For errors, an error message will be included to indicate the cause of the failure.

Code	Type	Reason
`201`	`CREATED`	Check successfully processed.
`400`	`BAD REQUEST`	There was a validation error with the request.
`500`	`INTERNAL ERROR`	There was a problem processing the request.

Errors

Error Messages

The following error messages may be returned:

Status Code	Type	Message	Description
`400`	`BAD REQUEST`	`Invalid transaction insight name: <...>`	An invalid transaction insight key was used. The invalid key is returned in the message.
`400`	`BAD REQUEST`	`Unsupported property name: <...>`	A valid property key was used, but the subscriber is not allowed to use it. The key name used is included in the message. Contact your Customer Success Manager or Integration Engineer to enable the attribute.
`400`	`BAD REQUEST`	`Rule not found for type <...>`	The type parameter in the request does not reference a valid ruleset. This occurs when the ruleset has not been created in the Business Rules Editor.
`400`	`BAD REQUEST`	`Invalid 'statedIp'`	The IP address provided as the statedIp in the request is not a valid IP address.

Status Code	Type	Message	Description
`401`	`UNAUTHORIZED`		The BASIC authentication credentials are incorrect or the subscriber id in the REST body is wrong.
`403`	`FORBIDDEN`		Your account does not have permission to use this API. Please contact your Integration Engineer for assistance.
`500`	`INTERNAL ERROR`		An error occured within the iovation service and the request was unable to be processed.

Example Error

```
{
  "path": "/v1/api/fraud/subs/1000/checks",
  "code": 400,
  "type": "BAD REQUEST",
  "message": "Invalid transaction insight name: 'goodPupper'"
}
```

Managing Evidence with the Evidence APIs

Overview

Use the evidence REST APIs to add, update, and retract evidence on devices and accounts.

HTTP Headers

To send an evidence request to iovation, you must include the `Content-Type` and `Authorization` HTTP headers.

Content-Type

You must set the `Content-Type` header in each request to `application/json`.

```
Content-Type: application/json
```

This is currently the only supported format.

Authorization

iovation uses *Basic Access Authentication* to authenticate API requests. Format user names as follows:

`<subscriber ID>/<subscriber account>`

For example:

`1000/OLTP`

Pass the user name and password via the `Authorization` header. The password is your subscriber passcode. The header value should be formatted as follows:

`Basic (Base64 encoded string of <user name>:<password>)`

For example:

```
Authorization: Basic MzM0NzAyL09MVFC6UFbCVvU1MkY=
```

Adding Evidence

Overview

Use the `evidence` endpoint to add new evidence to devices and accounts.

Resource URL

```
POST https://ci-api.iovation.com/fraud/v1/subs/{subscriberId}/evidence/
```

where *`subscriberId`* is your subscriber identifier.

Request Body

The request represents the evidence type and target using JSON. It should include the following parameters.

Parameter	Format	Description	Example
`evidenceType`	String	***Required*** The evidence type. You must submit the evidence type using the precise `X-X` format. For example, to submit evidence of credit card fraud, the format must be: `1-1` This can be any of the evidence types documented here: Evidence Types Reference. You can apply one instance of a given evidence type to an account. If you attempt to post the same type of evidence more than once, the existing evidence record will be updated with a new comment.	`3-9`
`comment`	String	***Required*** Comment describing the evidence. May be up to 4000 characters.	`Confirmed identity theft`
`appliedTo`	appliedTo entity	***Required*** The account or device to apply the evidence to. The `appliedTo` entity can contain an `account` or a `device`.	`"appliedTo": { "type": "account", "accountCode": "ai902gt0" }`

appliedTo Entity

The `appliedTo` entity represents which account or device to apply evidence to.

Parameter	Format	Description	Example
`type`	String [`account`|`device`]	***Required*** Defines the type of entity, either accounts or devices.	`account`
`accountCode`	String	***Required if*** `type` = `account` Account code to apply the evidence to.	`111702`
`deviceAlias`	String	***Required if*** `type` = `device` Device ID to apply the evidence to.	`370122194215027502`

Example Requests

The following request adds evidence type **1-4** to an account.

```
{
    "evidenceType": "1-4",
    "comment": "Charge-back from legit customer",
    "appliedTo": {
        "type": "account",
        "accountCode": "ai902gt0"
    }
}
```

The following request adds evidence type **1-4** to a device.

```
{
  "evidenceType": "1-4",
  "comment": "Charge-back from legit customer",
  "appliedTo": {
    "type": "device",
    "deviceAlias": "12331231242352454364"
  }
}
```

Response Format

The add evidence response includes the following properties in a JSON block:

Property	Format	Description	Example
`id`	UUID	Unique identifier for the evidence. This ID is necessary to update or delete the evidence later on.	`0909733d-b883-3172-bfac-da1616e525af`
`evidenceType`	String	The evidence type.	`1-4`
`comment`	String	The comment that is stored for the evidence.	`Chargeback from legit customer`
`appliedTo`	appliedTo entity	An entity that describes whether the evidence was applied to a device or an account, and provides the unique identifier for the device or account.	`"appliedTo": { "type": "account", "accountCode": "ai902gt0" }`

Add Evidence Response Header

Add evidence responses include the following location header, which is the path you can use to read / update / delete the evidence later on:

```
/fraud/v1/subs/{subscriberId}/evidence/{id}
```

Add Account Evidence Response Body Example

```
{
    "id": "0909733d-b883-3172-bfac-da1616e525af",
    "evidenceType": "1-4",
    "comment": "Charge-back from legit customer",
    "appliedTo": {
        "type": "account",
        "accountCode": "ai902gt0"
    }
}
```

Add Device Evidence Response Body Example

```
{
    "id": "0909733d-b883-3172-bfac-da1616e525af",
    "evidenceType": "1-4",
    "comment": "Charge-back from legit customer",
    "appliedTo": {
        "type": "device",
        "deviceAlias": "12331231242352454364"
    }
}
```

Response Codes

Code	Name	Reason
`201`	**Created**	The evidence was successfully created.
`400`	**Bad Request**	Any of the following: • `evidenceType` is set to an invalid evidence type; verify that this is set to one of the values listed in Evidence Types Reference . Evidence types must be set to the following format: `X-X`, for example `1-4`. • No comment was entered. • The `appliedTo` entity is missing or its values are not legitimate.
`404`	**Not Found**	The account or device specified by the `appliedTo` entity was not found in the system.
`409`	**Conflict**	This evidence type already exists.

Updating Evidence

Overview

You can update properties for existing evidence, including the comment and type.

Resource URL

```
PUT https://ci-api.iovation.com/fraud/v1/subs/{subscriberId}/evidence/{evidenceId}
```

where `subscriberId` is your subscriber identifier and `evidenceId` is the ID of the evidence that you added.

Request Body

Response Properties

The request represents the evidence type and target using JSON. It should include the following parameters.

Parameter	Format	Description	Example
`id`	String	***Required*** The unique identifier for the evidence that you will update. When you add evidence, this ID is returned in the response body in the `id` parameter.	`0909733d-b883-3172-bfac-da1616e525af`
`evidenceType`	String	Include this parameter if you want to modify the evidence type. You must submit the evidence type using the precise `X-X` format. For example, to submit evidence of credit card fraud, the format must be: `1-1` This can be any of the evidence types documented here: Evidence Types Reference.	`3-9`
`comment`	String	Comment describing the evidence.	`Confirmed identity theft`

Parameter	Format	Description	Example
`appliedTo`	appliedTo entity	The account or device to apply the evidence to. The `appliedTo` entity can contain an `account` entity or a `device` entity.	`"appliedTo": { "type": "device", "deviceAlias": "12331231242352454364" }`

Update Evidence Request Example

```
{
    "id": "0909733d-b883-3172-bfac-da1616e525af",
    "evidenceType": "1-4",
    "comment": "Updated evidence type",
    "appliedTo": {
        "type": "device",
        "deviceAlias": "12331231242352454364"
  }
}
```

Response Format

Response Properties

The update evidence response includes the following properties in a JSON block:

Property	Format	Description	Example
`id`	UUID	Unique identifier for the evidence.	`0909733d-b883-3172-bfac-da1616e525af`
`evidenceType`	String	The evidence type. This is always in the `X-X` format, such as `1-4`.	`1-4`
`comment`	String	The comment that is stored for the evidence.	`Chargeback from legit customer`
`appliedTo`	appliedTo entity	An entity that describes whether the evidence is associated with a device or an account, and provides the unique identifier for the device or account.	`"appliedTo": { "type": "device", "deviceAlias": "12331231242352454364" }`

Update Evidence Response Example

```
{
    "id": "0909733d-b883-3172-bfac-da1616e525af",
    "evidenceType": "1-4",
    "comment": "Updated evidence type to 1-4",
    "appliedTo": {
        "type": "device",
        "deviceAlias": "12331231242352454364"
    }
}
```

Response Codes

Code	Name	Reason
`200`	**Success**	The evidence was successfully updated.
`400`	**Bad Request**	The body of the request is incorrect.
`404`	**Not Found**	The evidence was not found.

Getting Evidence

Getting All Evidence From All iovation Subscribers (Consortium) for a Device or Account

Overview

This request returns a collection of evidence details associated with a specific device or account, including consortium evidence added by other iovation subscribers. The details, including evidence IDs, types, and comments, are returned in a JSON array.

Resource URL

```
GET https://ci-api.iovation.com/fraud/v1/subs/{subscriberId}/consortium/evidence?<request
parameters>
```

where *`subscriberId`* is your subscriber identifier and `<request parameters>` include one or both of the following:

Query Parameters	Value	Description
`accountCode`	String	Returns all evidence associated with an account. Make sure `accountCode` is properly URL encoded.
`deviceAlias`	String	Returns all evidence associated with a device.

You must include either `accountCode` or `deviceAlias` in your request, or you can specify both in order to get all evidence for both an accound and a device.

Request Example

The following example gets all evidence associated with the account `testUserName`, where the subscriber ID is `1000`:

```
https://ci-api.iovation.com/fraud/v1/subs/1000/consortium/evidence?accountCode=testUserName
```

The following example gets all evidence associated with the device `12345678`:

```
https://ci-api.iovation.com/fraud/v1/subs/1000/consortium/evidence?deviceAlias=12345678
```

Response Example

```
[
    {
        "evidenceType": "1-1",
        "source": "io_subscriber1",
    },
    {
        "evidenceType": "1-10",
        "source": "io_subsciber2",
    },
]
```

If no evidence is found, you will receive an empty array. If the same evidence type is assigned more than once, we only return it once.

Response Codes

Code	Name	Reason
`200`	**Success**	Evidence was successfully returned.
`400`	**Bad Request**	The request parameters were incorrect or insufficient.

Retracting Evidence

You can retract a single piece of evidence at a time, or retract a set of multiple pieces of evidence at a time.

Retracting a Single Piece of Evidence

You can specify a single evidence ID in order to retract it.

Resource URL

```
POST https://ci-api.iovation.com/fraud/v1/
subs/{subscriberId}/evidence/{evidenceId}/retractions
```

where `subscriberId` is your subscriber identifier and `evidenceId` is the ID of the evidence that you are retracting.

Request Body

The request body is a JSON block that includes a comment to describe the retraction.

Path	Value	Description
`comment`	String	**Required** Comment for the retraction.

Request Body Example

```
{
   "comment": "this is a retraction comment"
}
```

Response Body

No content is returned.

Response Codes

Code	Name	Reason
`204`	**No Content**	The evidence was successfully retracted.
`404`	**Not Found**	The specified evidence was not found.

Retracting All Evidence on a Device or Account

You can retract all evidence associated with a specific device or account. This is limited to devices and accounts known to you; it does not apply to other subscribers in the iovation consortium. You can limit this to a specific evidence type, or retract every type of evidence.

Resource URL

```
POST https://ci-api.iovation.com/fraud/v1/subs/{subscriberId}/evidence/retractions?<request
parameters>
```

where *`subscriberId`* is your subscriber identifier and `<request parameters>` is the account or device, and the type of evidence to remove.

Request Body

Query Parameters	Value	Description
`accountCode`	String	Retracts all evidence associated with an account. Make sure `accountCode` is properly URL encoded.
`deviceAlias`	String	Retracts all evidence associated with a device.
`evidenceType`	String	The type of evidence to retract.

You must include either `accountCode` or `deviceAlias` in your request.

Response Body

The response includes the number of pieces of evidence that were removed.

```
{
  "removedCount": 10
}
```

Response Codes

Code	Name	Reason
`204`	**No Content**	Evidence was successfully retracted.
`400`	**Bad Request**	The request parameters were incorrect or insufficient.

Code	Name	Reason
`404`	**Not Found**	No evidence of the type was found for the account or device.

Moving to Production

Overview

When the walk-through is complete, iovation begins the process of preparing the production environment.

Moving to Production

To move to production:

1. Provide pingable IP addresses to iovation so iovation can monitor network connectivity to between our data centers.
2. Update the following with production credentials from iovation:
    - Production URL for the REST or SOAP APIs
    - Subscriber id
    - Subscriber passcode
3. If your production (or test) environment blocks outbound requests, you must ensure outbound access to the following IPs:
    - For web service calls: 74.121.28.0/22 on port 443
    - If reverse proxying first party Javascript components:
        - 74.121.28.0/22 on port 443
        - 66.151.32.32/27 on port 443
        - 80.252.88.128/27 on port 443
4. Bookmark the URL to the production version of the iovation Intelligence Center and set up administration accounts.
5. Define and validate the production versions of your business rule sets.
6. Start sending production traffic.

Optimizing Live Integrations

Overview

After going live, iovation will work with you to optimize your integration.

Optimizing and Troubleshooting Live Accounts

After going live:

- iovation trains your team on best practices.
- Hold regular meetings with iovation team to discuss progress, answer questions, and tune business rules.
- Troubleshoot any transaction errors returned from iovation. For example, with the CheckTransactionDetails SOAP API:
  - If transactions receive "authentication denied" errors, double-check the credentials that are used by the SOAP requests. For example, ensure that correct production credentials, as opposed to test credentials, are in use. Also make sure that the correct SOAP URL is in use.
  - Provide the complete SOAP requests to your integration engineer for any transaction errors that you are unable to solve.
- Troubleshoot any transaction time-outs.

  Verify that the application server is connected to the internet and can connect to iovation. To do this, for the CheckTransactionDetails SOAP API, use:
  `telnet soap.iovation.com 443`
- Troubleshoot any HTTP errors returned from iovation. Submit the timestamp and requesting IP address to an iovation integration engineer.