Channel Partner Integration Guide

Channel Partner Integration Guide

Channel Partner Integration Guide

About Channel Partner Integration

Overview

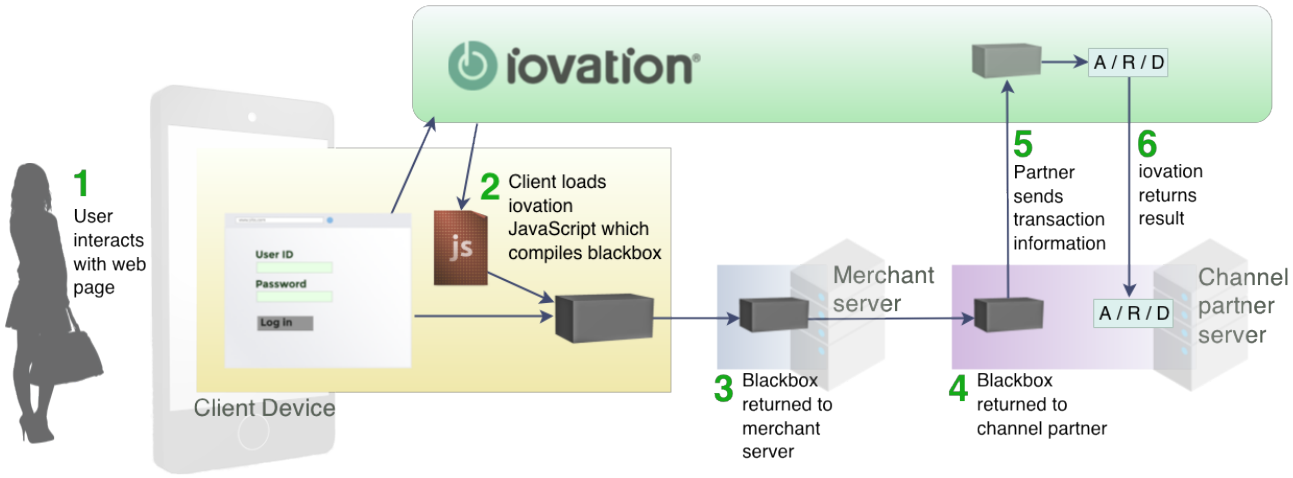
Explains what the feature is or what its benefits are to the user or customer.

Integration to Interaction Points

ReputationManager 360 recognizes devices as they attempt to interact with your services, and assesses risk associated with those devices. You can integrate ReputationManager 360 at any key points of end-user interaction such as account creation, login, updating account details, and checkout.

At each of these interaction points, ReputationManager 360 collects information about end-users' devices and creates a *blackbox*. A blackbox is an encrypted string that contains all of the device attributes that iovation is able to collect. The blackbox is passed, along with transaction details, to ReputationManager 360. ReputationManager 360 evaluates the transaction and, based on your business rules, returns a recommendation of *Allow*, *Review* or *Deny* along with details about the transaction. You must then choose how to respond to these results. For example, you might implement logic that automatically refuses access to your services when ReputationManager 360 returns a *Deny* result.

Reputation Check Process



Step	Description
1	The end user attempts to use your merchant's services.
2	The end user's browser loads JavaScript files from iovation. The JavaScript builds a blackbox.
3	The blackbox is returned to your merchant's server.
4	The blackbox is returned to your service.
5	Your server sends the blackbox, a unique account identifier, the user's IP address, transaction properties, and a business rule identifier to iovation using the *CheckTransactionDetails* SOAP API.
6	ReputationManager 360 processes business rules for the transaction type and returns a result of *Allow*, *Review*, or *Deny*.

Integrating to a Website

Overview

This topic explains how to integrate ReputationManager 360 into a website.

About Website Integration

Website integration includes the following steps:

1. Add JavaScript configuration parameters to define the end user experience and determine how device information is collected.
2. Include the iovation JavaScript on your page. This JavaScript will gather device information.
3. Return the collected device information, known as blackboxes, to your web server.

Integration Files and Requirements

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

iovation creates blackboxes using its own JavaScript, called *snare.js*. iovation dynamically generates snare.js whenever a user interacts with a page where you have included it. The JavaScript creates and stores a blackbox based on parameters that you have configured on your web pages.

Configuring the iovation JavaScript

Important Configuration Notes

- **You must place the configuration parameters before including the iovation JavaScript.** The configuration parameters determine the user experience and manner via which the blackbox is obtained. If the configuration variables are not defined prior to the JavaScript, the blackbox may not be retrieved or the user experience may not be what is desired.
- **Do not cache or use local copies of snare.js.** The iovation JavaScript is dynamically generated for every user; caching the script may cause unrelated devices to be identified as the same computer. The script also uses domain cookies to identify devices across subscribers.

Defining Configuration Parameters

Follow these steps to add a script tag to your page and declare configuration variables. These variables determine how the iovation JavaScript collects data and interacts with your end-users.

1. Add the following parameters to a `<script>` on your page. These should follow standard HTML script notation. For example:
2.
```
<script>
  var io_install_flash = false;
  var io_install_stm = false;
</script>
```

Configuration Value	Type	Default Value	Description
`io_install_flash`	boolean, optional	`false`	Determines whether the user is prompted to install or upgrade Flash if the required version of Flash is not installed. If the required version of Flash is installed, this setting has no effect.
`io_flash_needs_handler`			If `io_install_flash` is set to false, this handler will not run. Users may see an error if the required version of Flash is not installed to their systems. Use this variable to define your own JavaScript error handling for this condition. For example, you can define your own error message: `var io_flash_needs_update_handler = "alert( 'Install Flash!' );";`
`io_install_stm`	boolean, optional	`false`	Determines whether the user is prompted to install the iovation Active X control. The Active X control helps collect hardware information. This control is only available for Internet Explorer. If the Active X control is installed, this setting has no effect.
`io_exclude_stm`	bitmask, optional	`15`	Determines whether to use the iovation Active X control, if it is installed. When Active X is available in Internet Explorer (version 8+ and on Windows Vista), security warnings may appear when the control is installed but not yet run. You can opt to disable the control for specific platforms and therefore prevent the prompt from occurring. This bitmask has the following set of values: ◦ **`0`** – default (run on all platforms) ◦ **`1`** – Do not run on Windows 9x (including Windows 3.1, 95, 98 and Me) ◦ **`2`** – Do not run on Windows CE

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

Including the iovation JavaScript

To include the iovation JavaScript, add the following `<script>` element to your web page. Note that this is the URL to the version of snare.js in the iovation production environment.

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

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

Troubleshooting Errors

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

This is a global JavaScript variable; you can review it with any JavaScript that runs after the iovation JavaScript has loaded. For example, use the following snippet:

```
<script>
alert("Last captured iovation error: " + io_last_error);
</script>
```

Collecting Blackboxes

After the iovation JavaScript creates a blackbox, you must collect it and send it back to your server. Your server will then send it to iovation to process.

Methods for Collecting Blackboxes

There are three ways to collect a blackbox.

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

Implementing the Hidden Form Field Collection Method

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

Implementation Steps

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

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

2. In the JavaScript configuration parameters, set the `io_bbout_element_id` parameter to the id of the hidden form field. For example, if the `id` attribute is set to `ioBlackBox`, set `io_bbout_element_id` to the following:
`var io_bbout_element_id = 'ioBlackBox';`
3. In the JavaScript configuration parameters, set the parameters as needed for the user experience you want to set up. For example:

- Set the `io_install_flash` and `io_install_stm` parameters to `false` to ensure that end users are not prompted to update their Flash and Active X installations if they do not have the required versions installed.
- Set `io_exclude_stm` to `12` to prevent the iovation Active X control from running on either Windows Vista or Windows XP.
- Set `io_enable_rip` to `true` to collect Real IP information.

Hidden Form Field Collection Example

Here is a sample web page with full integration code in place.

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

<body>
    <div id="main">
        <form action="/do_ctd" method="post" name="loginSubmitForm" id="loginSubmitForm">
            <fieldset>
                <ul>
                    <!-- Create hidden field 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>
    </div>
    <!-- These variables execute before the JavaScript files load. They set default values for the
JavaScript. -->
    <script>
        var io_bbout_element_id          = 'ioBlackBox';
        var io_install_stm               = false;
        var io_exclude_stm               = 12;
        var io_install_flash             = false;
        var io_enable_rip                = true;
    </script>
<!—Include JavaScript library -->
<script type="text/javascript" src="https://mpsnare.iesnare.com/snare.js">
</script>
</body>
</html>
```

Implementing the Callback Function Collection Method

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

Implementation Steps

1. In the JavaScript configuration parameters, set the `io_bb_callback` parameter to a function that processes the blackbox value, and that has the following signature:
   `function bb_update_callback( bb, complete)`
   Where:
   - `bb` is the updated value of the blackbox
   - `complete` is a boolean value that indicates whether all collection methods (Flash, Active X etc.) are complete

   NOTE If `io_bb_callback` and `io_bbout_element_id` are both specified, the hidden field specified in `io_bbout_element_id` will not be populated, unless explicitly done so by the function specified in `io_bb_callback`.
2. Set the JavaScript configuration parameters as needed. For example:
   - Set the `io_install_flash` and `io_install_stm` parameters to `false` to ensure that end users are not prompted to update their Flash and Active X installations if they do not have the required versions installed.
   - Set `io_exclude_stm` to `12` to prevent the iovation Active X control from running on either Windows Vista or Windows XP.
   - Set `io_enable_rip` to `true` to collect Real IP information.

Callback Function Collection Example

For an example, see the next section on the `ioGetBlackbox` collection method.

Implementing the ioGetBlackbox Collection Method

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

Implementation Steps

To implement the `ioGetBlackBox` collection method:

1. Call the function. For example:

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

   This returns an object with the following attributes:
   - `blackbox` – the updated value of the blackbox
   - `finished` – a Boolean indicating whether all the collection methods have completed.
2. In the JavaScript configuration parameters, set the parameters as needed for the user experience you want to set up. For example:
   - Set the `io_install_flash` and `io_install_stm` parameters to `false` to ensure that end users are not prompted to update their Flash and Active X installations if they do not have the required versions installed.

- Set `io_exclude_stm` to `12` to prevent the iovation Active X control from running on either Windows Vista, Windows 7, or Windows 8.
- Set `io_enable_rip` to `true` to collect Real IP information.

Callback and ioGetBlackbox Collection Example

If you do not have a form or if you need to control other aspects of the user interactions when the blackbox is available, you can combine the callback and `ioGetBlackbox` methods to establish a maximum wait time.

This example illustrates how you can use these two methods together to transmit the blackbox to your server via an AJAX call, and wait a maximum of 5 seconds before getting the available information.

```
>// basic configuration
var io_install_stm = false;   // do not install Active X
    var io_exclude_stm = 12;      // do not run Active X
    var io_install_flash = false;   // do not install Flash
    var io_enable_rip = true;     // collect Real IP information
    var sent = false;

function send_bb( bb ) { // function to process the blackbox
    if ( sent ) return;
        // process blackbox, this might consist of making an ajax call enabling
        // a submit button or submitting the form
        // $.ajax({ type: 'POST',
        //          url: 'blackbox-collector-url',
        //          data: 'blackbox:'+blackbox,
        //          dataType: 'text'
            //         });
            sent = true;
            }

// use callback to process blackbox as soon as it is done
    var io_bb_callback = function (bb, isComplete)
    { // when done call the send function
        if ( isComplete ) send_bb( bb );
    };

// set up timer to give us a maximum wait time before giving up on waiting
    // for all of the data to be collected. At that point, just send what is
    // available
        setTimeout( function() {
            try {
            var bb_info = ioGetBlackbox();
            send_bb( bb_info.blackbox );
            return;
            } catch (e) {}
        send_bb( “” );     // if we are done but got an error,
                                // send an empty blackbox
        }, 5000);  // set a maximum wait time of 5
</script>
<!—Include JavaScript library -->
<script type="text/javascript" src="https://mpsnare.iesnare.com/snare.js">
</script>
```

Integrating into Mobile Apps

Overview

This topic explains how to integrate the iovation Mobile SDKs into your iOS and Android apps.

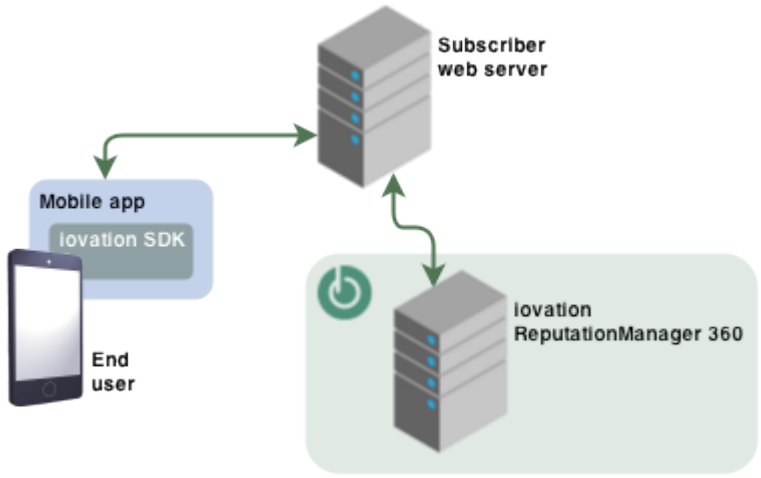
Downloading iovation Mobile SDK for iOS and Android

If you have not already downloaded the iOS or Android SDKs, you must do so before proceeding. Log into the iovation Help Center via the Administration Console, and download the files from here:

http://help.iovation.com/Downloads

About Mobile Integration

Add the iovation Mobile SDK to your apps to collect information about your end-users' devices. The client generates a blackbox that contains all available device information.



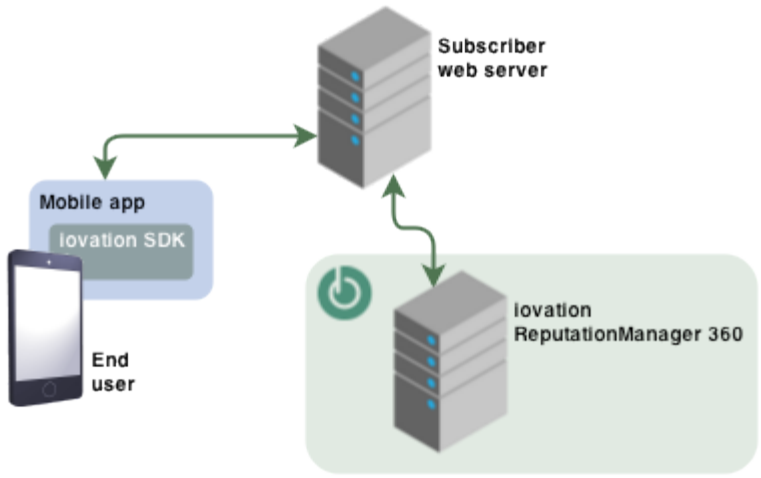
Integrating with iOS Apps

Overview

Complete these steps to integrate the iovation SDK for iOS with your apps.

About Mobile Integration

Add the iovation Mobile SDK to your apps to collect information about your end-users' devices. The client generates a blackbox that contains all available device information.



iOS Integration Files and Requirements

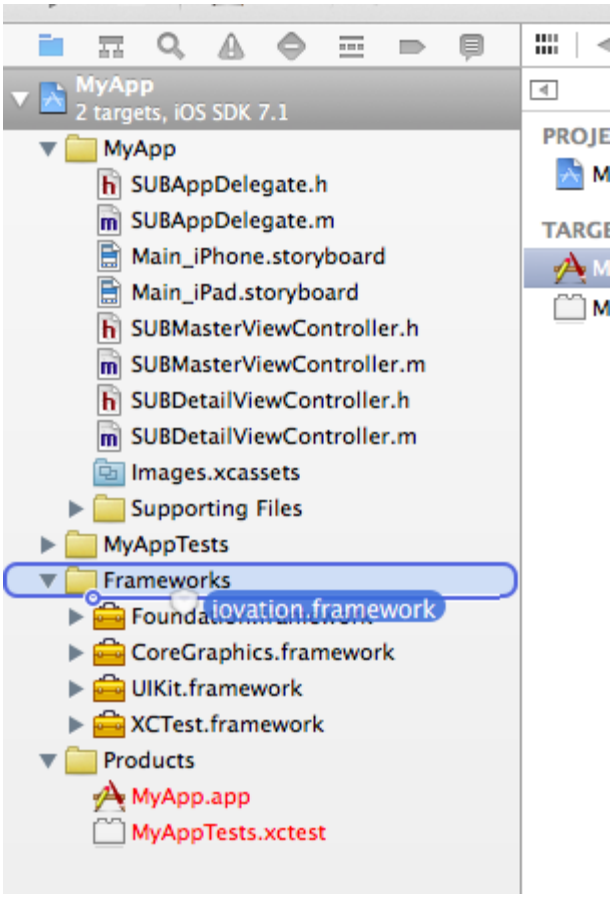
File	*iovation.framework*
Version	3.1.0
Required frameworks	• *CoreTelephony.framework* • *ExternalAccessory.framework*
Optional Frameworks	• *AdSupport.framework* • *CoreLocation.framework*

Version 3.1.0 of the iovation iOS SDK supports iOS 5.1.1 or higher on the following devices:

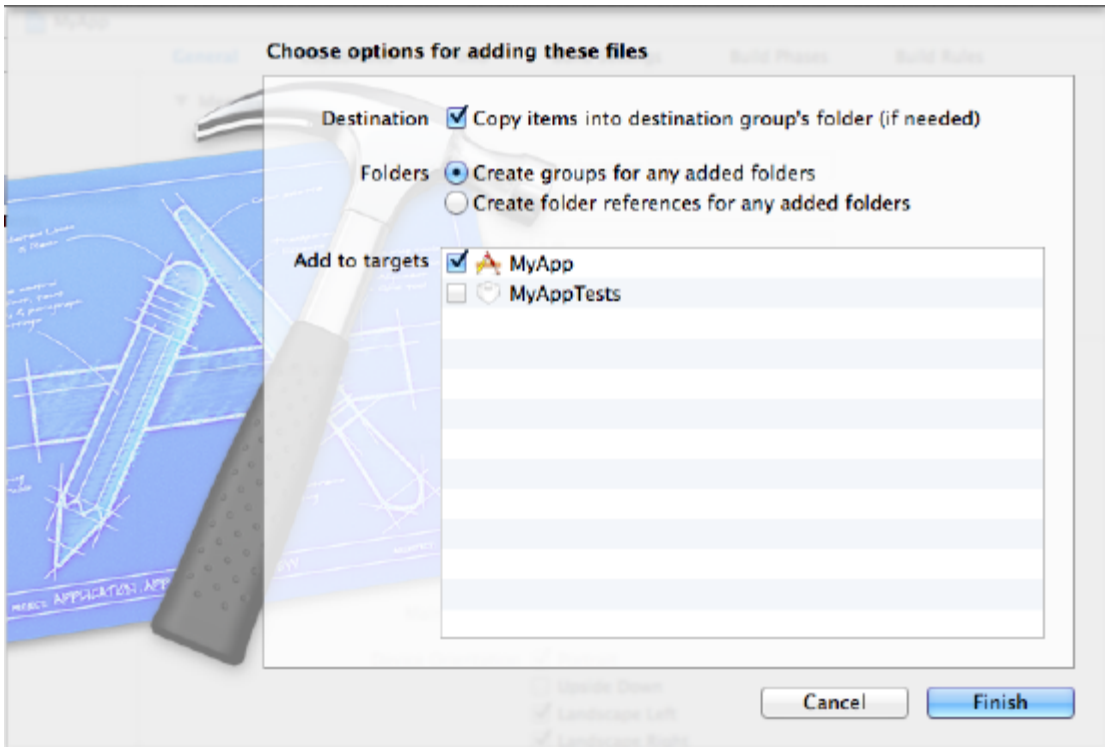
- iPhone 3GS and later
- iPod Touch 3rd generation and later
- All iPads

Installing the iovation Mobile SDK for iOS

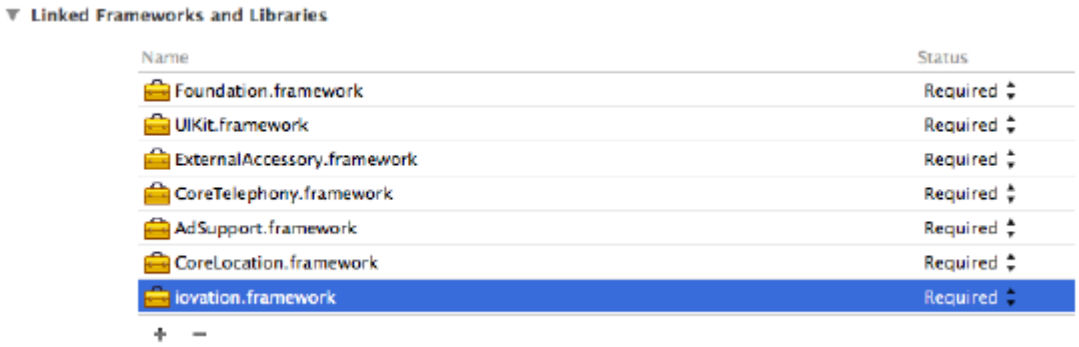
1. Download and unzip the SDK. See
2. In Xcode, drag *iovation.framework* into your project's navigation area.
3. Select **Copy items if needed** to copy the framework file into your project's directory.



4. In the dialog that appears:
   - Select **Copy items if needed** to copy the framework file into your project's directory.
   - Check the checkbox for the targets in which you plan to use the framework.



5. Click **Finish**.
6. Add the following frameworks to your application's target in XCode:
   - *ExternalAccessory.framework*.If you find that the Wireless Accessory Configuration capability is turned on in Xcode 6 or higher and you don't need it, turn it off and add *ExternalAccessory.framework* again.
   - *CoreTelephony.framework*.



7. Optionally add these frameworks if your app makes use of them:
   - *AdSupport.framework* If your app displays ads. **Do not include if your app does not use the ad framework. 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 application requests geolocation permission from the user.**

Using the +ioBegin Function

The `+ioBegin` function collects information about the device and generates a blackbox that contains this information. Submit this string in an HTTP request to your service. See the iovSample Xcode project included in the download for a sample implementation.

Syntax

```
NSSstring *bbox = [iovation ioBegin];
```

Return Values

`bbox` - a string that contains a blackbox.

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

Example

```
#import "sampleViewController.h"
#import <iovation/iovation.h>

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

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

@end
```

For a more extensive example, including submitting a blackbox in an HTTP request, see the iovSample Xcode project included in the download.

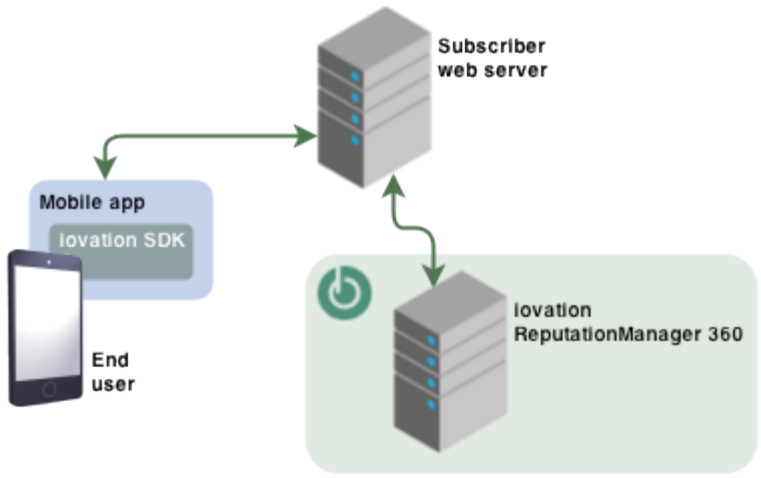
Integrating with Android Apps

Overview

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

About Mobile Integration

Add the iovation Mobile SDK to your apps to collect information about your end-users' devices. The client generates a blackbox that contains all available device information.



Android Integration Files and Requirements

SDK Filename	deviceprint-lib-1.2.0.aar
Version	1.2.0
Package	com.iovation.mobile.android.DevicePrint
Android SDK dependencies	Android SDK 2.1 or higher (SDK level 7)
Required Permissions	INTERNET
Optional Permissions	• BLUETOOTH • ACCESS_WIFI_STATE • READ_PHONE_STATE • INTERNET • 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.

Version 1.2.0 of the iovation Mobile SDK for Android supports Android 2.1 or higher.

Installing the SDK for Android

1. Download and unzip deviceprint-lib-1.2.0.aar.
2. Launch your IDE of choice.
3. In Eclipse and Maven, deploy the .aar file to your local Maven repository, using maven-deploy. http://maven.apache.org/guides/mini/guide-3rd-party-jars-local.html

4. In Android Studio, select File -> New Module. Expand *More Modules* and choose *Import existing .jar or .aar package*.
5. Select the deviceprint-lib-1.2.0.aar file, and press **Finish**.
6. Ensure that the deviceprint-lib is a compile-time dependency in the build.gradle file.

```
dependencies {
    compile fileTree(dir: 'libs', include: '*.jar')
    compile project(':deviceprint-lib-1.2.0-SNAPSHOT')
}
```

Using the ioBegin Function

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

Syntax

```
public static String ioBegin(Context context)
```

Parameters

`context` – An instance of the *android.content.Context* class used to access information about the device.

Return Values

A string that contains a blackbox.

IMPORTANT! The blackbox returned from `ioBegin` should never be empty. An empty blackbox or a blackbox that contains only `0500` indicates that the protection offered by the system may have been compromised.

IMPORTANT! You must package the deviceprint-lib-1.2.0.aar file with the application.

Compiling the Sample App in Android Studio

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 to your target Android SDK version.

1. Download and unzip deviceprint-lib-1.2.0.aar
2. In Android Studio, select File | Open or click **Open Project** from the quick-start screen.
3. From the directory where you unzipped deviceprint-lib-1.2.0.aar, open the android-studio-sample-app directory.
4. Open the *DevicePrintSampleActivity* file.
5. With some configurations, Android Studio may detect an Android Framework in the project and not configure it. In this case, open the Event Log and click **Configure**.
6. A pop-up prompts you to select the Android framework. Click **OK** to fix the framework errors.
7.

In Android Studio, select File -> New Module. Expand *More Modules* and choose *Import existing .jar or .aar package*.

8. Select the deviceprint-lib-1.2.0.aar file, and press **Finish**.
9. Ensure that the deviceprint-lib is a compile-time dependency in the build.gradle file.

```
dependencies {
    compile fileTree(dir: 'libs', include: '*.jar')
    compile project(':deviceprint-lib-1.2.0-SNAPSHOT')
}
```

10. Open the DevicePrintSampleActivity folder.
11. In the project navigation view, open src/main/java/com/iovation/mobile/android/sample/DevicePrintSampleActivity.java
12. Right-click the file editing view and select *Run DevicePrintSampleAct*.
13. Select either an attached physical device, or an Android virtual device to run the app on.
14. The app should now compile and launch.

Example Android App

The following example app assumes a simple view-based application where a button click populates a label containing the value of the blackbox.

For a much richer example, see the Android Studio sample app included with the SDK.

```
package com.iovation.mobile.android.sample;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import com.iovation.mobile.android.DevicePrint;
public class DevicePrintSampleActivity extends Activity {
    /**
     * Called when the activity is first created.
     * @param savedInstanceState If the activity is being re-initialized after
     * previously being shut down then this Bundle contains the data it most
     * recently supplied in onSaveInstanceState(Bundle). <b>Note: Otherwise it is null.</b>
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void printDevice(View target) {
        String bb = DevicePrint.ioBegin(getApplicationContext());
        TextView bbResult = (TextView) findViewById(R.id.bbResult);
        bbResult.setText(bb);
        bbResult.setVisibility(View.VISIBLE);
    }
}
```

Integration FAQ

Can I check account or device status programmatically without sending a transaction?

No. You can manually check account and device status in the Administration Console, but all programmatic checks are recorded as transactions.

Do you offer a REST version of your interface?

No. At this time ReputationManager 360 offers a SOAP interface.

Can we collect the blackbox elements on our own and pass them to you?

Not at this time. By using and managing dynamic javascript, iovation can update it as needed and keep pace with the rapid evolution of the internet.

We don’t control any web pages ourselves, our customers do; how should we handle the Javascript?

You can include the iovation Javascript in an iframe or in your own Javascript file and have your customer include that.

Is it possible to have a smartphone make the SOAP call directly using the iPhone and Android SDKs?

In order for subscribers' systems to communicate in real time with the iovation production environment, iovation registers IP address ranges for each subscriber and provides access to the iovation firewall to those systems. iovation does not, and cannot, allow direct access from your customers' devices. iovation does offer a light weight Javascript-only integration method - called a gateway integration - where the Javascript includes a wrapper function that enables one-way reputation checks as well as device printing. Under this model, the client device sends information directly to iovation servers and there is no real-time response. You are instead notified of results via reporting.

What is an API?

An API (application programming interface) provides a way for one software program to communicate with another software program. This makes it possible to share data and features between programs.

What is a Gateway Integration (AFA)?

A gateway integration allows device and account information to be sent directly to iovation from the end user's computer. This integration method requires no change to business logic. There is no real-time response back to the subscriber. Instead, notification of fraud or suspicious behavior is sent via reports or e-mail alerts. Subscribers must manually process the response.

What is a SOAP Integration?

Simple Object Access Protocol (SOAP) is a mechanism for communication between software applications on the Web. In an iovation SOAP integration, the subscriber sends ReputationManager 360 an account code, IP address, and device information via a SOAP call. ReputationManager 360 responds in real-time with a result.

How long does it take to implement ReputationManager 360 after the contract is signed?

Implementation time depends on resources available within your company. A gateway integration typically takes about two weeks and a SOAP integration takes up to one month, including testing.

Is there an API that can return the latest account and device status without sending a transaction?

You can check status manually using the Administration Console. Any programmatic checks via the iovation APIs are transactions and incur charges.

We use Amazon Web Services (AWS) to host our services. What IP should we give iovation for firewall access?

For our subscribers who use AWS or other cloud based services, we recommend setting up a proxy server and then providing us with the IP address for the proxy server.

Why am I not getting a blackbox?

3rd-party JavaScript must be enabled in end-users' browsers in order for snare.js to run and generate blackboxes. Some browser plugins and proxy services block 3rd party JavaScript.

Verify the following:

1. For web integration, verify that the iovation JavaScript **snare.js** is included just one time on the page; multiple inclusions can cause unwanted interactions between the two copies.
2. If you are using the hidden form field method, the `io_bbout_element_id` parameter is defined and the hidden field exists on the page.
3. If you are using the callback function method, the `io_bb_callback` parameter is defined and functions.
4. Check the value of the `io_last_error` variable to see if any JavaScript errors are generated while the collection process is running. You can also check the browser's error console to see if snare.js generated any errors.

Is a form required?

You do not need a form. You can use io_bb_callback or ioGetBlackbox to retrieve and process blackboxes as needed.

Can I host my own local copy of snare.js or cache it?

snare.js is dynamically created for each user. This allows iovation to include network information in a blackbox. The iovation hosted copy also uses iovation domain cookies to help correlate devices across all of iovation's subscribers.

As a result, a locally hosted or cached copy would not support sharing of information across iovation subscribers and all devices would be identified with the same device id.

I have multiple forms that need a blackbox, how do I support this?

The delay mechanism will only submit one form - which may not be the one the user requested to be submitted

You can handle this problem through the use of `io_bb_callback`:

```
<html>
<head>
<script type="text/javascript">

var io_bb_callback = function (bb, complete) {
               // populate both form fields
               signupform.ioBB.value = bb;
               loginform.ioBB.value = bb;
             };
</script>
</head>

<body>
  <!—include field that will contain blackbox and configure snare.js ?
     <script language="javascript">
     var io_enable_rip              = true;
     var io_flash_needs_update_handler = "";
</script>
    <script type="text/javascript" src="https://mpsnare.iesnare.com/snare.js"></script>

    <!—first form definition ?
      <form id="loginform" name="loginform" method="post" action="#" onsubmit="return validate_bb(
this );">
     <input type="text" name="ioBB" id="ioBB" value="" />
     <input type=submit>
     </form>

    <!— second form definition ?
     <form id="signupform" name="signupform" method="post" action="#" onsubmit="return validate_bb(
this );">
     <input type="text" name="ioBB" id="ioBB" value="" />
     <input type=submit>
     </form>
</body>
</html>
```