PRO•D
Remote SDK

White-Label Integration Guide

Including:

PRO•D Application SDK

- **iOS** [SDK ver. 2.28]
- **Android** [SDK ver. 1.24]

PRO•D Cloud API [ver. 1.1.0]

TABLE OF
CONTENTS

Introduction to PRO•D White-Label Developer Guide

This document will explain simple methods of creating mobile apps that can perform software and hardware tests on the mobile device (the phone running the app) and present the results on the device or submit to a centralized cloud-services database for reporting and analysis purposes.

To learn about creating integration to our Cloud-Services API to access results of Diagnostics and statistical data.

There are two components of the PRO•D white-label solution that are explained in this document: the Application SDK and the Cloud-Services API.

What is PRO•D Application SDK?

In brief, PRO•D Application SDK is a set of software functions that are created for Android and iOS devices, to perform tests at the lowest level of the operating system software accessing hardware and electronic components of the device.

What is PRO•D Cloud-Services API?

The PRO•D Cloud-Services API manages the coordination of sessions. To begin a Remote Diagnostic, a **`SessionID`** must be created using the cloud-services API and shared to the white-label Application using the integrated application SDK.

Once the remote diagnostics are complete, the application SDK automatically send the results to the cloud-services for storage.

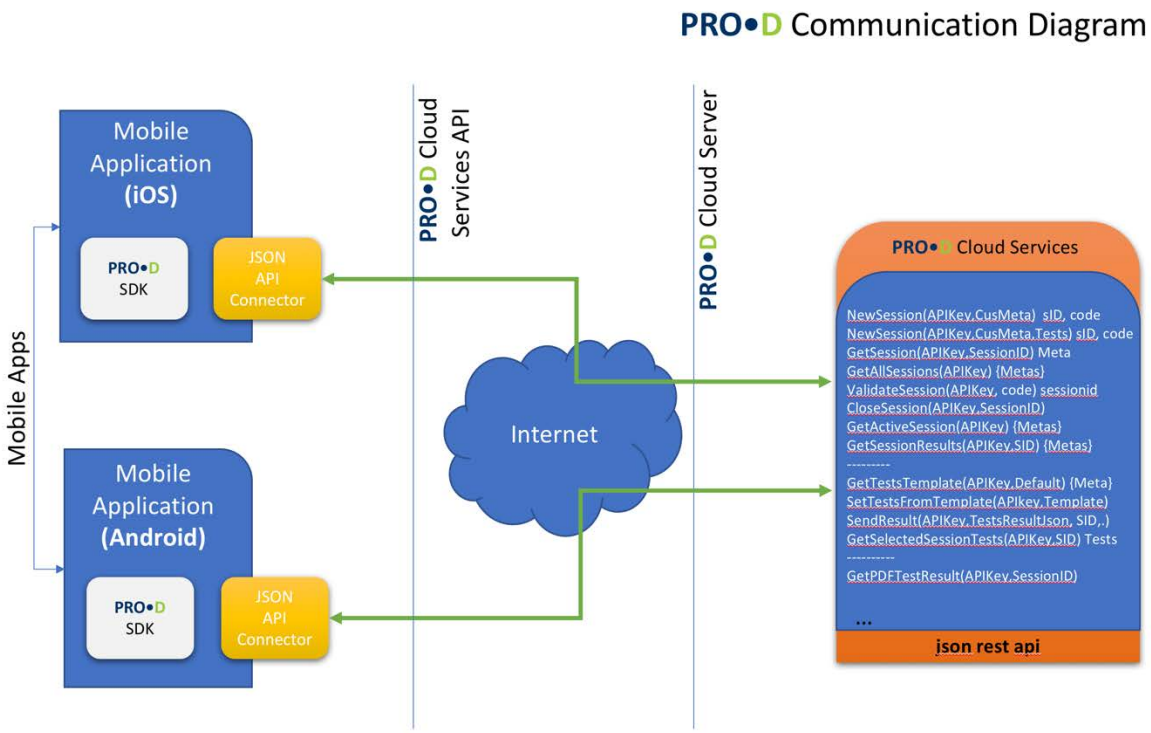
Your 3rd party solution can use the cloud-services API to initiate sessions, query active sessions activity, query results of completed sessions, etc.

How does the PRO•D Solution Function?

To use the PRO•D solution to start testing devices, an application that can run on the device has to be created using the provided application SDK for either of the platforms (iOS, Android) and installed on the device. At the same time the PRO•D cloud-services are integrated to actively collect and store the test result data produced by the device.

Additionally, it is also possible to display the test results on the device screen.

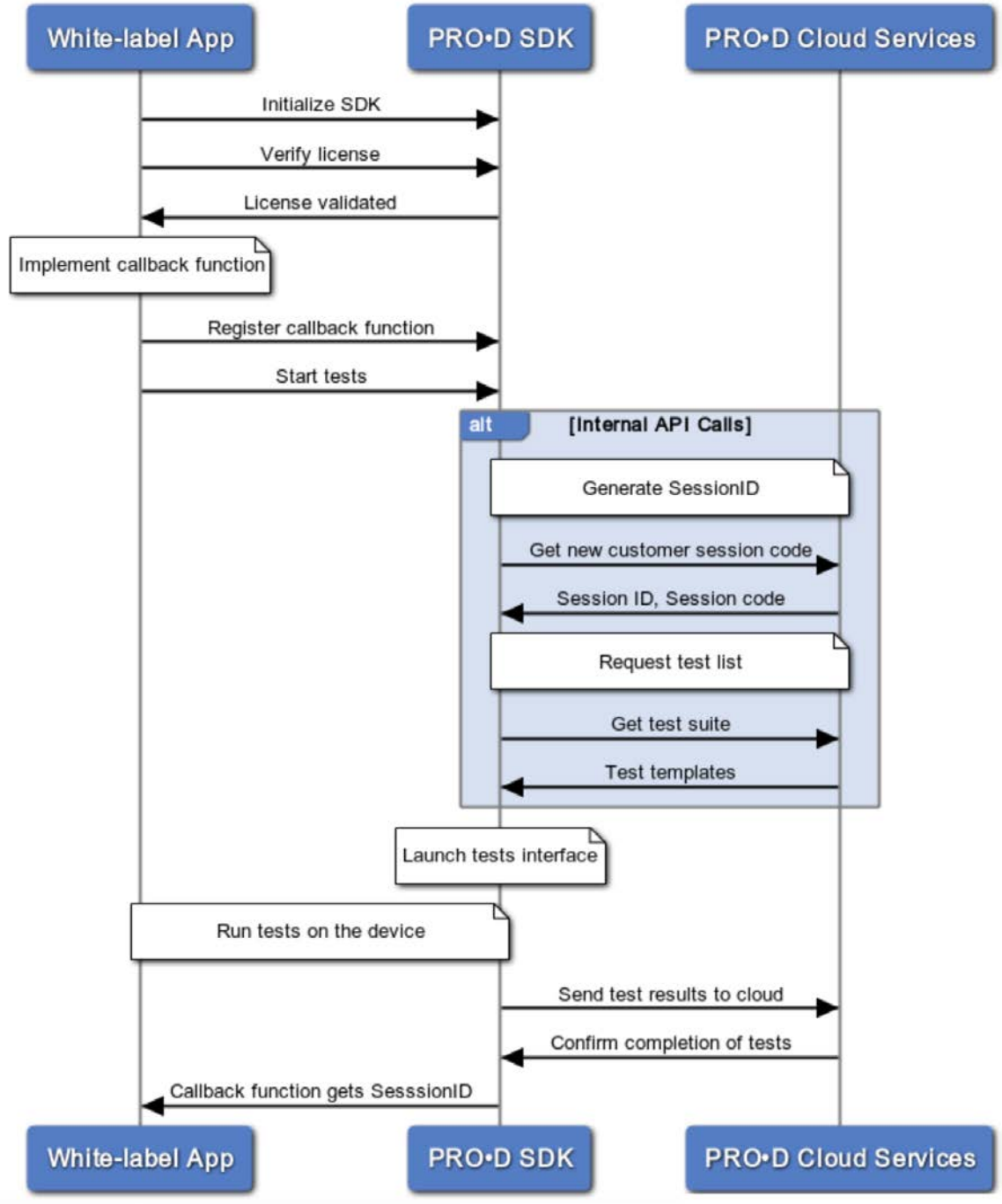
The following diagram briefly represents the communication between the mobile device using the Application SDK and transferring the data to the Cloud via the cloud-services API:



At the completion of the tests, the results of the tests can be retrieved in the JSON format by making calls to the Cloud-Services API.

On-device Tests Sequence Diagram

The following diagram shows the communication stages between the White-label App, the PRO•D SDK, and PRO•D Cloud-Services API in the proper stages of execution:

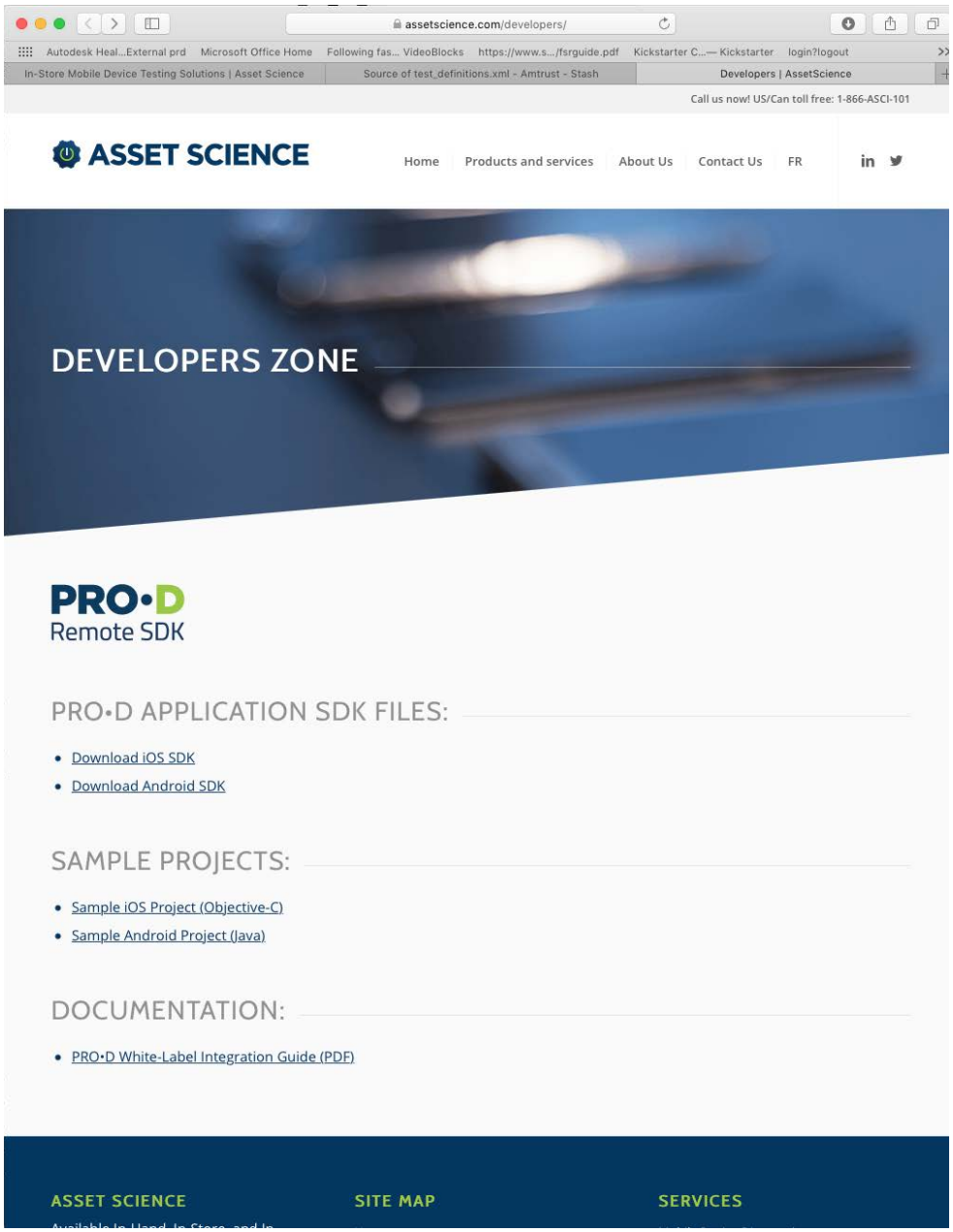


The name of the only callback function is **`diagnosticSessionCompleted().`**

Creating the Mobile App for testing the device

To ease the development process, Asset Science has provided some sample projects that are very easy to customize by the developers. These white label template projects can be found on the *Developers Zone* page and can be used as a starting point. The online location of the Developers Zone page on the Asset Science website is:

http://developers.assetscience.com



There are different template projects for different platforms supported (iOS and Android). They are created using the common development tools for each platform, Xcode for iOS, and Android Studio for Android. These sample projects are written in their framework's popular programming languages, Objective-C for iOS and Java for the Android platform.

They simply start the proper SDK version, create a connection with the server and make calls to the PRO•D Cloud-Services API to receive a Session ID and register the device. After opening a session with the server, then they start making some basic tests and submitting the results to the server. The complete list of test functions and API calls can be found at the bottom of this document.

Rebranding PRO•D Mobile App

Each one of these projects allow the developers to quickly make necessary changes to the resources that are used for branding, such as logos and color themes. Internal text fields and variables are listed in the **ReadMe** files and are very straight forward to find and modify, to customize the app.

Rebranding can be done as simple as making modifications to some fields in the source code or the configuration files, e.g. such as the following block:

```
{
    "General": {
        "vendor": "ProDSDK App Demo",
        "vendorHexColor" : "003366",
        "vendorHighlightHexColor":"FFFFFF"
    }
}
```

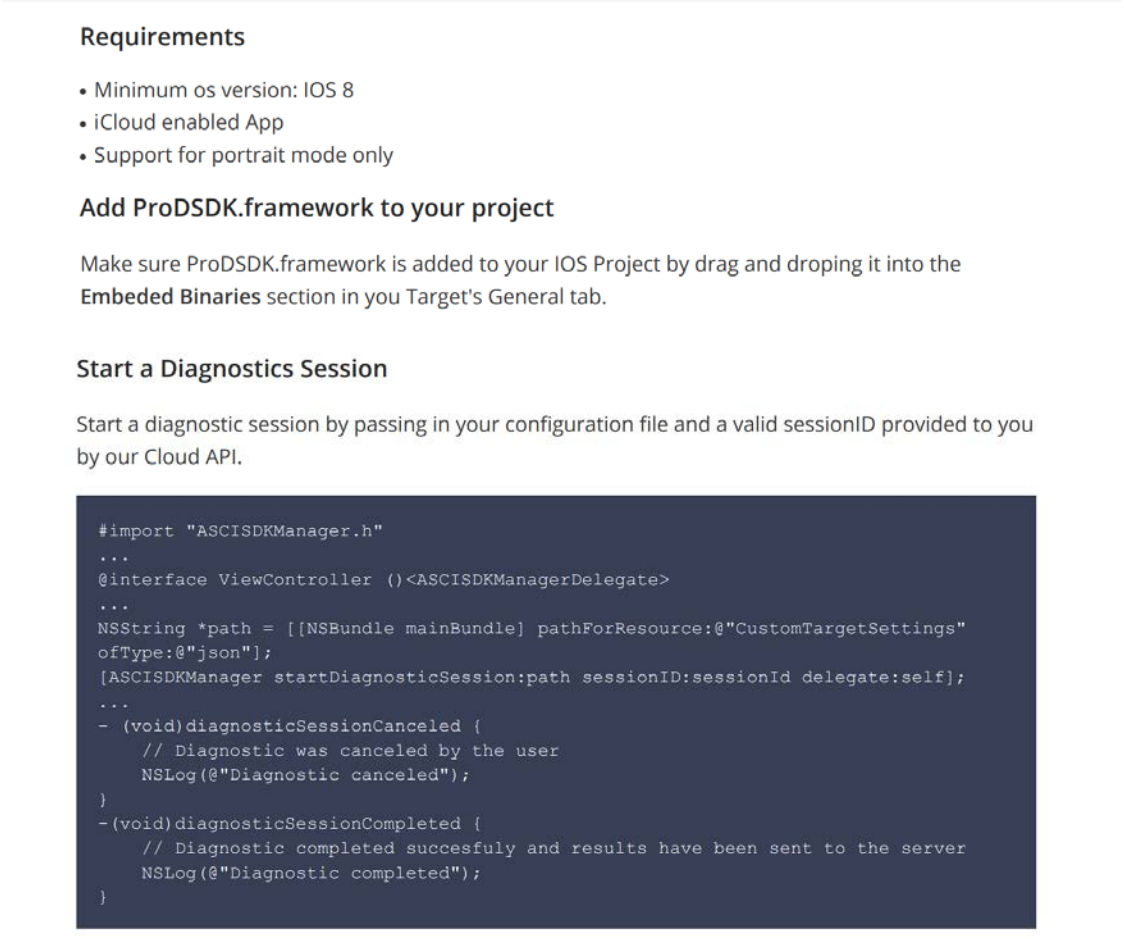
Configuration of Test Suites to Run on the Device

A complete list of tests is available and can be used to build proper test suites for the White-label App, in order to specify the tests to run on the device. The test suite configuration details and tools can be obtained by getting in touch with Customer Success team at Asset Science.

PRO•D Application SDK Quick Start (iOS)

The following source code snippets show how simple is to make calls to the Application SDK to perform a test on the device. There are common sections between both supported platforms, but there are also minor platform-dependent differences in some of the SDKs functions, that are explained and addressed in the template projects and can be re-used in the final app with minimal or no modifications.

Note: *The following snippets and other contents are also available in the Developer Zone page on the Asset Science website in their respective sections.*



Requirements

- Minimum os version: IOS 8
- iCloud enabled App
- Support for portrait mode only

Add ProDSDK.framework to your project

Make sure ProDSDK.framework is added to your IOS Project by drag and droping it into the **Embeded Binaries** section in you Target's General tab.

Start a Diagnostics Session

Start a diagnostic session by passing in your configuration file and a valid sessionID provided to you by our Cloud API.

```
#import "ASCISDKManager.h"
...
@interface ViewController ()<ASCISDKManagerDelegate>
...
NSString *path = [[NSBundle mainBundle] pathForResource:@"CustomTargetSettings"
ofType:@"json"];
[ASCISDKManager startDiagnosticSession:path sessionID:sessionId delegate:self];
...
- (void)diagnosticSessionCanceled {
    // Diagnostic was canceled by the user
    NSLog(@"Diagnostic canceled");
}
-(void)diagnosticSessionCompleted {
    // Diagnostic completed succesfuly and results have been sent to the server
    NSLog(@"Diagnostic completed");
}
```

Setting up your project

You need to add a few keys to your Target's **info.plist**

```
    <key>NSCameraUsageDescription</key>
    <string>Used to test camera hardware</string>
    <key>NSMicrophoneUsageDescription</key>
    <string>Used to test microphone hardware</string>
    <key>NSExceptionDomains</key>
    <dict>
        <key>pdm-qa.asci.io</key>
        <dict>
            <key>NSExceptionAllowsInsecureHTTPLoads</key>
            <true/>
            <key>NSIncludesSubdomains</key>
            <true/>
        </dict>
    </dict>
```

You will need a configuration file. This is where you set parameters for things like color for theme and wich server to connect to. More configurations to come in the future

```
{
    "General": {
        "vendor": "ProDSDK App Demo",
        "vendorHexColor" : "003366",
        "serverUrl" : "http://pdm-qa.asci.io/apidevices"
    }
}
```

Notice that the **serverUrl** parameter must match the **NSExceptionAllowsInsecureHTTPLoads** property in the info.plist

CocoaPods

Our SDK requires 3rd party libraries to be loaded in your project. We recommend using CocoaPods. Here is a sample Podfile content with the required libraries

```
use_frameworks!
target 'ProDSDK-App-2' do
    platform :ios, '8.0'
    # Needed for Reachability (WiFiConnectivity test, etc.)
    pod 'Reachability'
    # Needed for logging
    pod 'CocoaLumberjack', '1.9.2'
    # Needed for XML parser
    pod 'KissXML'
    # Needed for the DST Export to keep keys in order
    pod 'OrderedDictionary', '1.1.1'
    # Needed to unzip testSuite file
    pod 'SSZipArchive'
end
```

PRO•D Application SDK Quick Start (Android)

The following steps are to perform tests on Android platform the same way that are explained in the previous section for iOS, and the template projects and can be re-used in the final app with minimal or no modifications.

Add Library Module to your project

Make sure the aar library is added to your Android Project following these steps. Note in the future, this library will be hosted somewhere, and will not need to be added manually using this method.

Import the Library Module

Add the library as a dependency in your app's build.gradle:

```
compile project (':prodiagnosticsSDK')
```

In the current version of the SDK, you will have to import the dependencies listed below. In the next version, this will not be necessary.

```
compile 'com.android.support:support-v4:23.4.0'
compile 'com.android.support:appcompat-v7:23.4.0'
compile 'com.android.support:support-v13:23.4.0'
compile 'com.android.support:multidex:1.0.1'
compile 'com.google.code.gson:gson:2.3.1'
compile 'com.google.protobuf:protobuf-java:2.6.1'
compile 'com.davemorrissey.labs:subsampling-scale-image-view:3.1.3'
compile 'com.google.guava:guava:18.0'
compile 'com.github.gabrielemariotti.cards:cardslib-core:2.1.0'
compile 'com.sun.mail:android-mail:1.5.5'
compile 'com.sun.mail:android-activation:1.5.5'
compile 'com.amazonaws:aws-android-sdk-s3:2.2.15'
compile 'com.github.paolorotolo:appintro:4.1.0'
compile 'com.android.support:design:23.4.0'
compile 'com.squareup.retrofit2:retrofit:2.2.0'
compile 'com.squareup.retrofit2:converter-gson:2.2.0'
compile 'com.squareup.okhttp3:logging-interceptor:3.6.0'
compile 'com.android.support.constraint:constraint-layout:1.0.2'
```

Initialize the Diagnostics Library

The SDK must first be initialized in your Application class.

```
    public class MyApplication extends MultiDexApplication {
        @Override
        public void onCreate() {
            super.onCreate();
            ProDiagnostics.init(this);
        }
}
```

Start a Diagnostic Session

Once the SDK is initialized, start a diagnostic session by passing in your Activity's context and a valid sessionID provided to you by our Cloud API.

```
   myButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ProDiagnostics.startDiagnosticSession(YourActivity.this,
"38c242ec-58fa-4afb-afe8-b0bad899b28c");
            }
        });
}
```

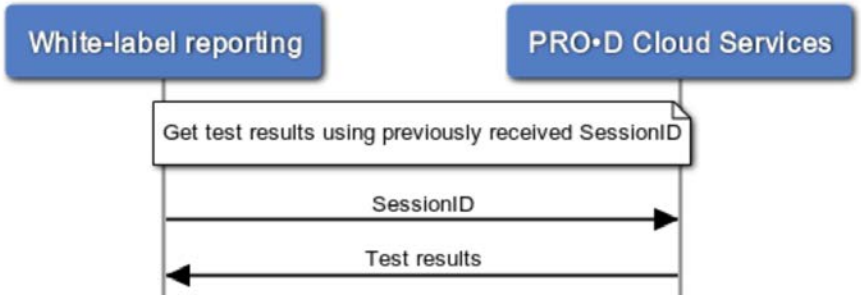
Completing Diagnostics

Once the diagnostic process has completed on the device, the library will close and return to your application. In upcoming versions of the SDK, we will likely have a callback method that you can register for to listen for this kind of event. For now, you can check the status of the device diagnostics when your own application's Activity resumes.

```
    @Override
    protected void onResume() {
        super.onResume();
        if (ProDiagnostics.getInstance().hasDiagnosticSessionCompleted()) {
            // Do what you want here
        }
    }
```

Retrieving Test Results using PRO•D Cloud-Services API

Upon completion of the diagnostics on the device, the results can be queried using the PRO•D Cloud-Services API, passing in the same **SessionID** that was received from the Application SDK on the initialization method call (the initial method is called **startDiagnosticSession()** on both iOS and Android platforms).

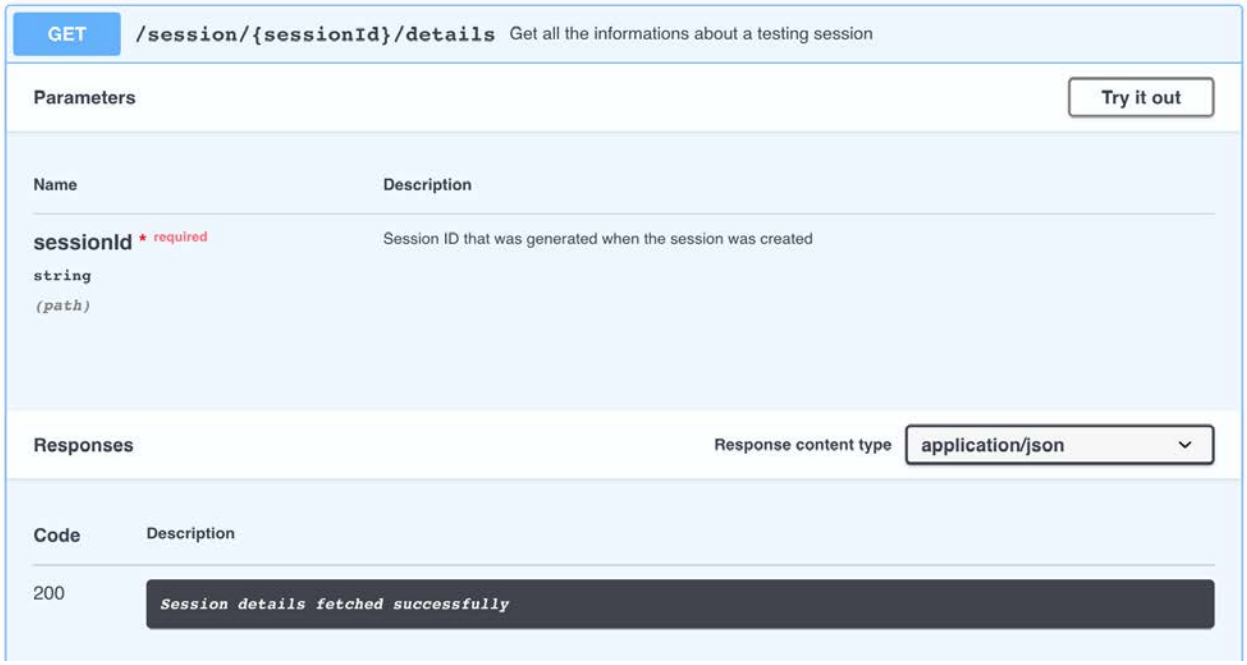


RESTful call to the PRO•D Could-Services API

The following call to Cloud-Service API (Restful) with the required **SessionID** as the only parameter, will return the results of the diagnostics for the device mapped to the specified **SessionID**.

The Root URL to be used with all API calls is:

https://consumermanagement.qa.asci.io



Example Value | Model


```
{
  "success": true,
  "message": "string",
  "payload": {
    "session": {
      "license": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0",
      "sessionId": "a1a1a1a1-a1a1-a1a1-a1a1-a1a1a1a1a1a1",
      "code": "a1a1a1",
      "callbackUrl": "http://api.customer.com/test-results-callback",
      "metadata": {
        "myImportantId": 1234,
        "anArrayOfThings": [
          "thing1",
          "thing2"
        ]
      },
      "createdOn": "2018-01-12T14:07:30.753-05:00",
      "closedOn": "2018-01-12T14:07:30.753-05:00",
      "runs": [
        {
          "createdOn": "2018-01-12T14:07:30.753-05:00",
          "tests": [
            "TC-00001"
          ],
          "completedOn": "2018-01-12T14:07:30.753-05:00",
          "results": {},
          "resultsJson": "{\"recipient\":\"fleclerc@assetscience.com\",\"deviceInfo\":
{\"modelNumber\":\"SM-GN950\",\"identifiers\":[{\"name\":\"IMEI\",\"value\":\"123456789\"},{\"name\":\"Ser-
ial number\",\"value\":\"SER102939848575\"}],\"storage\":\"32 GB\"},\"reportDetails\":
{\"startTimestamp\":\"2017-10-31 11:41:09\",\"endTimestamp\":\"2017-10-31
11:48:09\",\"result\":\"FAIL\"},\"testResults\":[{\"name\":\"VolumeUpButtonTest\",\"result\":\"PASS\"},
{\"name\":\"FlashTest\",\"result\":\"FAIL\"},{\"name\":\"VideoRecordingTest\",\"result\":\"PASS\"},
{\"name\":\"MenuButtonTest\",\"result\":\"FAIL\"},
{\"name\":\"WiFiConnectivityTest\",\"result\":\"PASS\"}]}"
        }
      ]
    },
    "testsNames": {
      "EdgeScreenPaintTest": "Edge Screen Painting",
      "FlashTest": "Flash LED",
      "LcdPaintTest": "Touch Screen",
      "MenuButtonTest": "Menu Button",
      "VideoRecordingTest": "Video Recording",
      "VolumeUpButtonTest": "Volume Up",
      "WiFiConnectivityTest": "Wi-Fi Connectivity"
    }
  }
}
```


404

Session not found. No session were found for the given session id.

PRO•D Cloud-Services API Documentation

The following are the Restful API definitions that make the server side functionality available to the Application SDKs running on the devices.

Note: The details of the following API can be found in the Developer Zone section of the Asset Science's Control-Center Portal.

The root URL is:

https://consumermanagement.qa.asci.io

Consumer Management

[Base URL: 127.0.0.1:4450]

Consumer Management Service.

Customer application

GET /sessions/{license}
Get a list of all the session for a given license

GET /sessions/{license}/active
Get a list of all the active session for a given license

POST /session/{variable}
Creates a new device testing session *variable is a license*

DELETE /session/{variable}
Close a device testing session *variable is a sessionid*

PUT /session/{variable}
Add a suite of device tests to a session *variable is a sessionid*

GET /session/{sessionId}/details
Get all the information about a testing session

GET `/session/{sessionId}/metadata`
Get the metadata that was supplied when the session was created.

GET `/tests`
Get a list of all the device tests available, sorted by category

GET `/tests/{platform}`
Get a list of all the device tests available for the given platform, sorted by category

Device application

POST `/session/{variable}/results`
Send the device tests results to the session. *variable is a session code*

GET `/session/{code}/config/{platform}`
Get the list of tests to run on the device from the session