Factsheet

Agent SDKs – iOS SDK

iOS SDK

iOS SDK

The SamKnows iOS SDK and its plugin frameworks allow you to leverage the SamKnows suite of tests within your own app with minimal setup. You can select from any of the currently supported tests (see Available Tests). Along with the test-specific metrics, we also collect various environmental and connection metrics (see Data Collection Specifics) and submit this to the SamKnows One platform, along with your panel key, ensuring availability of your data. It is also possible to specify another URL to submit the metrics to and this can be set via the SKDMetrics object. Please see the API documentation for specifics on this.

SDK concepts

The information here provides a high-level break down of the key parts of the SDK. The included sample app shows use of these parts too. However, for a more in-depth explanation, please consult the docs supplied with the SDK release.

Contents of the SDK:

docs –
API documentation for the SDK
Frameworks - Frameworks required to build apps that use the SDK
SKDarwin_Samples.xcworkspace – Example app showing how to use the SDK
README.md - This document detailing how to get set up with the SDK

The SamKnows iOS SDK contains a number of plugin frameworks that are contain wrappers around the same tests as used on the various SamKnows hardware platforms. To run a test, you first need to create an instance of the test's TestRunner implementations. When creating one of these test runner instances, you provide a RunnerCallbacks, which defines a closure to be called when the test completes and an optional progress closure, that will be called at a frequency specified within the struct.

Most tests need a URL to connect to. If you use a single endpoint, this is a straight-forward string to supply on creation of the tests. However, if you run multiple hosts and want the tests to run against the most opportune server at that time, you may find the TestHostLocator a useful tool, as it will get a list of potential hosts and determine the ideal one to use for the currently running test(s).

During test execution, you will receive an instance of TestResult with each progress report and another when the test completes. There should never be a requirement to instantiate this class directly, but knowledge of its existence and the properties contained within the various test-specific results is good to have.

While the tests can be run directly with the TestRunner class, it is advisable to make use of the TestingAgent class, passing it a SKDarwinTestSequence and letting the agent take care of the rest. You can optionally inject a TestingAgentDelegate to the testing agent and in turn, will be notified of progress through the test sequence. It's worth noting that the TestingAgent will work equally well with a SKDarwinTestSequence of a single, or multiple tests.

Requirements

The SamKnows iOS SDK supports the two most recent major releases of iOS (currently iOS 10 and iOS 11) and is built with the current release version of Xcode. The SDK may work with older versions and beta versions of iOS, but is untested.

Available tests

The SamKnows iOS SDK currently supports five tests. These are Download, Upload, Jitter, Webget and YouTube.

Download/upload

These tests can be run by creating either a DownloadTestRunner or UploadTestRunner as required. The documentation for these tests can be found in the following directory: ./docs/SKDHttpTest/Classes/HTTPTestRunner.html. Returns a HTTPTestResult, for documentation: ./docs/SKDHttpTest/Classes/HTTPTestResult.html.

Jitter (packet loss, latency)

This test can be run by creating a JitterTestRunner. The documentation for this test can be found in the following directory: ./docs/SKDJitterTest/Classes/JitterTestRunner.html. Returns a JitterTestResult, for documentation: ./docs/SKDJitterTest/Classes/JitterTestResult.html. The JitterTestResult also contains roundTripTime for Latency, and packetsSent and packetsReceived, enabling the user to calculate the percentage of lost packets.

Webget

These tests can be run by creating a WebgetTestRunner. The documentation for these tests can be found in the following directory: ./docs/SKDWebgetTest/Classes/WebgetTestRunner.html. Returns a WebgetTestResult, for documentation: ./docs/SKDWebgetTest/Classes/WebgetTestResult.html.

YouTube

These tests can be run by creating a YouTubeTestRunner. The documentation for these tests can be found in the following directory: ./docs/YouTubeTest/Classes/YouTubeTestRunner.html. Returns a YouTubeTestResult, for documentation: ./docs/YouTubeTest/Classes/YouTubeTestResult.html.

Obtaining the SDK

The SDK is available to licenced users only and is delivered via a secure SamKnows FTP server. If you are already licenced to use the SDK, contact your SamKnows account manager to obtain the credentials to download it.

Measurement servers

The measurement clients in the SamKnows SDK carry out their measurements against dedicated measurement servers provided by SamKnows. Some examples of these are included in the sample app. When using the SDK, you will need to configure a list of measurement servers for it to use.

A list of dedicated measurement servers appropriate for your geography and use-case can be obtained from SamKnows. Alternatively, you may wish to host your own measurement servers. In this instance, SamKnows can provide a software package suitable for installation on RHEL/CentOS 6.x or 7.x servers. For more information about measurement servers please contact your SamKnows account manager.

To find the best server for use, consider using the TestHostLocator. Documentation can be found in ./docs/SKDarwin/Classes/TestHostLocator.html. Providing a URL with a formatted file at the endpoint will determine which element from the list is the most suitable host at the time.

Data collection

The SamKnows iOS SDK reports its measurements to one or more data collection endpoints. Data is transmitted to the collection endpoint in JSON format over an HTTPS connection. The data collected is

limited to the measurement results themselves, environmental information (e.g. Wi-Fi and cellular connectivity status) and a persistent unique identifier for the handset. No personal information is collected. By default, the SDK is configured to transmit measurement results to the SamKnows data collection endpoint exclusively.

Data collection specifics

The following environmental information is collected:

Field	Description
agentID	From the server, after being linked to a panel.
deviceID	Generated on the device and saved to the keychain.
locationData	This is added if the user has granted the application location access. This needs to be requested in the application itself, the SDK does not request permissions. Note that this contains horizontalAccuracy, verticalAccuracy, latitude, longitude, and locationAddress.name and locationAddress.postalCode if the last known placemark was able to be found.
manufacturer	This is always set to "Apple" for iOS.
deviceModel.	Contains the model code.
osName	System name.
osVersion	Version of the system.
osDisplayFull	String in the format: "System name system version"
memoryUsed	Assigned memory is how much is currently reserved for the app's sandbox. May not explicitly reflect the current physical footprint.
memoryFree	Available memory is the amount of memory not currently in use by this app, the system, or other apps.
indoorState	Whether the user is indoors, outdoors, or unknown

The following connection information is collected:

Field	Description
connectionType	wi-fi or cell.
cellularTechnology	2G/3G/4G
carrierName	String containing the name of the subscriber's cellular service provider.
carrierNetworkCode	String containing the mobile network code for the subscriber's cellular service provider, in its numeric representation
carrierCountryCode	String containing the mobile country code for the subscriber's cellular service provider, in its numeric representation
ssid	Wi-Fi network name.

Additional data collected:

Field	Description
appVersion	The version of the application the user is running..

All of this data is stored against the panelKey you provide, as well as being submitted to SamKnows One.

Sample metrics submission

The following is an example of the JSON submitted when a test sequence is complete. This sample includes just the data from a latency/jitter test. Note that the data outside of tests is included once per submission, but the tests/<test name>/environment object is captured and included within each test object for that sequence.


```
{
    "version": 1,
    "agentId": "<id assigned by sk1>",
    "data": {
    "device_environment": {
    "carrier_name": "EE",
    "id": "<auto-generated device id>",
    "mobile_network_code": "30",
    "iso_country_code": "gb",
    "operating_system_version": "iOS 11.2.2",
    "mobile_country_code": "234",
    "model": "iPhone10,6",
    "manufacturer": "Apple"
    },
    "metadata": {
    "app_version": "v7 b2"
    },
    "tests": {
    "latency": {
    "round_trip_time": 99817,
    "mean_opinion_score": 1,
    "stream_bits_per_sec": 51200,
    "utc_datetime": "2018-01-22T13:43:52Z",
    "target": "n1-palermo-it.samknows.com",
```

```
    "failures": 0,
    "local_datetime": "2018-01-
22T13:43:52Z",
    "environment": {
    "other": {
    "is_logged_in": false
    },
    "location": {
    "lon": 0,
    "lat": 51,
    "accuracy": {
    "horizontal": 65,
    "vertical": 10
    },
    "address": "135 Park Street, SE1 9EA"
    },
    "telephony": {
    "cellular_technology": "4g",
    "connection_type": "cell"
    },
    "memory": {
    "device_free": 1796,
    "app_used": 95,
    "device_used": 0
    }
    },
    "status_code": 0,
    "packet_size": 160,
    "successes": 1,
    "packets_received": 200,
    "duration": 5005490,
    "packets_sent": 200,
    "jitter": 28491
    }
    }
    }
    }
```


Sample apps

Included with the SamKnows iOS SDK is an Xcode workspace called SKDarwin_Samples. Open this to see two sample apps; one written in Swift and one with Objective-C. The apps themselves are simple in their function; they list the included tests and below said list is a button. Pressing this button will run the tests and their results are output to the list below the button.

While the Obj-C project is very minimalist, with all the relevant code placed in ViewController.m you will notice that the Swift sample contains explanations about we're doing and why. The initialisation of panel id is shown in AppDelegate and the bulk of the test sequence and launch is contained within ViewController. The sample also includes a class called HTTPCallbacks which serves as an example of how you might want to encapsulate the test progress/completion callbacks when designing your app. It also includes inline creation of callbacks for the non-http tests, as either pattern is valid. Finally, the class ExampleDelegate shows an example implementation of the TestingAgentDelegate protocol. This can be used to be informed of overall progress and events occurring while a test sequence is running.

Installation

Drag and drop:
With SamKnows iOS SDK downloaded and decompressed, navigate to the Frameworks directory. Contained within this directory are all the frameworks required to run the tests your organisation wishes to use. Move these frameworks into the appropriate location of your project and then from here, drag and drop them into the project's Embedded Binariessection within the General tab of the project settings viewer.

Getting started

Setup:
There are a couple of values you may wish to set. While it is not a requirement that you do this in the AppDelegate class, it is common practice to do so. Begin by importing the module import SKDarwin. Next, set the values as desired. For example, in the AppDelegate's didFinishLaunching:

```
func application(_ application: UIApplication,
didFinishLaunchingWithOptions
launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) ->
Bool {

    SKDTestMetrics.initialiseWith(panelKey:
<your panel key>)
```

```
    SKDKeychainConfig.group =
"your.keychain.group"

    return true
    }
```

The panel key will be an organisation-specific ID assigned when signing up to the platform. If your app makes use of shared keychain access, you may wish to set the group's identifier to the SKDKeychainConfig.group.

Creating a test sequence

At its core, SKDarwinTestSequence is an alias for an array with elements of type TestRunner and so elements can be added the same way they would be for an array. Below, we create a test sequence and then add a test to it.

```
import SKDarwin
    import SKDHttpTest

    // ...

    var testSequence =
SKDarwinTestSequence()

testSequence.append(DownloadTestRunner(
withHost: "test.myhost.com", onPort: 9999,
progressCallbacks: httpCallbacks))
```

Note that in the above snippet, as well as importing SKDarwin, we've also put SKDHttpTest as we're using the Download HTTP test. If using other tests, it would be necessary to import the module for that test too. If you're using the SDK with Objective-C, note that instead of import FrameworkName you'll instead follow the usual Swift-ObjC interop pattern such that the above imports would be:

```
#import <SKDarwin/SKDarwin-Swift.h>
    #import <SKDHttpTest/SKDHttpTest-
Swift.h>
```

Running the tests

With a test sequence defined, whether loaded from file, hard-coded, or user defined, we can run the tests by passing the test sequence to a TestingAgent either at creation, or when running the tests themselves. While the example below shows the creation of the agent, it will need to be retained for at least the duration of the tests.

```
// Passing test sequence on creation:
    let agent = TestingAgent(forTestSequence:
testSequence)
    try? agent.run() // This function will throw
an error if attempting to run a test sequence
without one being present.

    // Passing test sequence as part of call to
run:
    let agent = TestingAgent()
    agent.run(testSequence:
self.testSequence) // Function doesn't throw,
as a test sequence must be provided as part
of call.
```

As the tests will take a large amount of time, they are run in the background. With this in mind, if you intend to perform any UI updates in reaction to test operation, be sure to do so on the main thread.

Objective-C settings

In addition, for Objective-C projects, ensure that under Build Settings tab, the option of the Build Options: Always Embed Swift Standard Libraries should be set to Yes.

That's it!

At the minimum, that is all that is required for running a test, or sequence of tests. As mentioned, there are other ways of doing the same as above, though this is the easiest. In addition, by implementing TestingAgentDelegate you will have access to auxiliary information about the running of the tests.

It is definitely worth having a look at the sample app and reading the accompanying documentation.

Releasing

To archive your application for release on the AppStore, you'll need to strip the debug symbols from the framework. This can be

accomplished by adding the following script to your Build Phases:

```
APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"

    # This script loops through the frameworks embedded in the application and
    # removes unused architectures.
    find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
    do

FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist" CFBundleExecutable)

FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
    echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"

    EXTRACTED_ARCHS=()

    for ARCH in $ARCHS
    do
    echo "Extracting $ARCH from $FRAMEWORK_EXECUTABLE_NAME"
    lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o "$FRAMEWORK_EXECUTABLE_PATH-$ARCH"

EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
    done

    echo "Merging extracted architectures: ${ARCHS}"
    lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
    rm "${EXTRACTED_ARCHS[@]}"

    echo "Replacing original executable with thinned version"
    rm "$FRAMEWORK_EXECUTABLE_PATH"
    mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"

    done
```