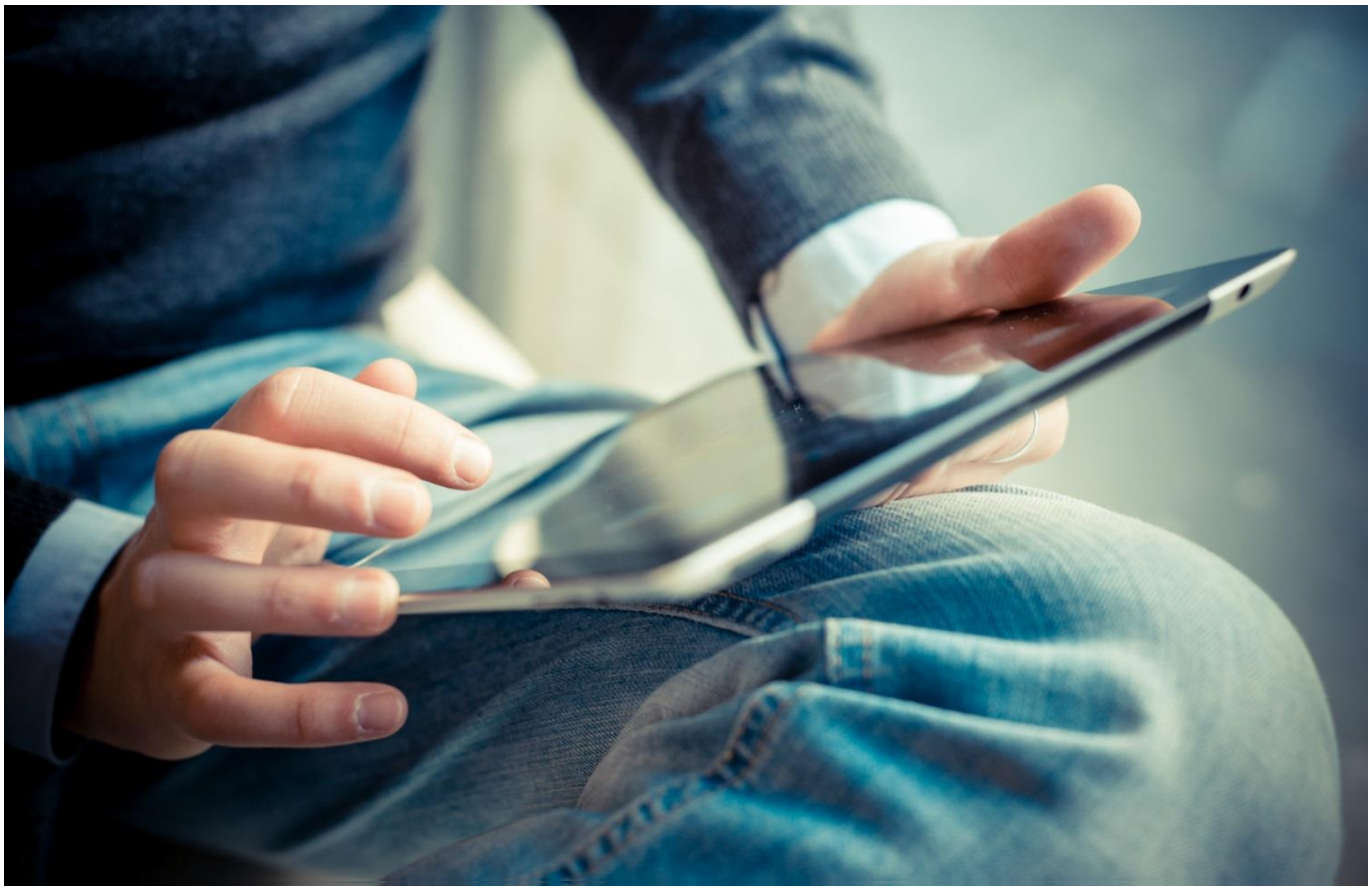


SENSIC STREAM AGENT API

Implementation guide for tracking video usage (iOS)

PLaNET PBI GfK

Last change: 27.11.2015

Table of contents

1 Introduction

1.1 About Digital Audience Measurement (DAM) with SENSIC in Poland

The motivation behind SENSIC DAM is to offer a platform infrastructure to track census-based data for the DAM product via GfK's SENSIC technologies. The publishers of web content (web sites / mobile apps / video streams) implement tracking tags / tracking SDKs – such as the SENSIC Stream Agent - into their web content to identify them via project + media + content identifiers and classify the content based on a taxonomy for post analysis.

SENSIC DAM provides several tracking libraries for the most used content types and platforms, which are embedded into the web content and send information to the SENSIC DAM tracking infrastructure once a web page is requested or upon certain events in apps / media players. These tracked impressions are processed, validated and enriched with additional information before stored in a distributed storage cluster.

1.2 About this document

This document contains **the documentation of the SENSIC Stream Agent Library for Apps ("SSA-A")**, that is used to measure the usage video content within an iOS app. This guide will provide you an overview of how the API for iOS looks like and how it should be implemented.

2 How to integrate

To track video stream usage with SSA, a tracking library has to be integrated into the app code and bound to the events of the used media player.

Some parameters for the tracking logic need to be set by the publisher along with the embedded library code. Other relevant information is set automatically by the SSA tracking logic.

The following sections describe the technical integration of the SSA-A library in general. Implementing and using the SENSIC Stream Agent library for iOS apps requires iOS 7.

2.1 Integration of the SSA-A library

The first thing you need to do is to integrate the SSA-A library to your app code.

To do so, drag & drop the folder `SSA-SDK` into the Xcode project navigator view of your app.
The folder should contain the following files:

- `SSA-SDK/libSSA.a`
- `SSA-SDK/SSA.h`

Click the project in the Project Navigator, click your app's target, then build settings, then search for "Other Linker Flags", click the + button, and add '-ObjC'.

2.2 Instantiate SSA

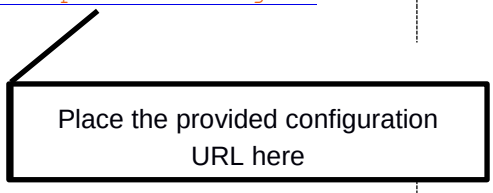
To work with the SSA library, you need an instance of the SSA class. Even though there can be multiple instances of this class, usually there should not be a need for creating more than one instance of it.

Instantiate the framework with `[[SSA alloc] initWithAdvertisingId:@"..." andConfigUrl:@"..."]` when the app starts. The provided `advertisingID` is used to identify the user. Note: If no `advertisingID` or `configUrl` is supplied, then the SSA will not track any events. The configUrl is the URL to the configuration file provided by GfK.

You can use the `AppDelegate` class for this. Here is an example:

```
#import "AppDelegate.h"
#import "SSA.h"
#import "Constants.h"
#import <AdSupport/ASIdentifierManager.h>


@implementation AppDelegate


SSA *SSA = nil;
// some methods and fields omitted for brevity
```

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
   // ...
    NSString *idfaString = [[[ASIdentifierManager sharedManager]
advertisingIdentifier] UUIDString];

    if (idfaString) {
        SSA = [[SSA alloc]
                initWithAdvertisingId: idfaString
                andConfigUrl: @"https://config.sensic.net/pl1-ssa-ios.json"
               ];
    }
    else {
        // do not track
        SSA = [[SSA alloc] init];
    }

    return YES;
}
// ...
```



2.3 Generate tracking requests

2.3.1 Acquire an Agent

To track the user's view time you need to acquire an agent. To do so, you have to call the `getAgent` method of the `SSA` class and provide it your *MediaID*. In this case there will be exactly one instance of an agent per *MediaID*.

The usage is shown in the following snippet:

```
#import "VideoPlayerController.h"
#import "SSA.h"

// some imports omitted for brevity

extern SSA *SSA;

@interface VideoPlayerController () {
        Agent *agent = nil;
    }
    // some properties omitted for brevity
@end

@implementation VideoPlayerController

// some methods omitted for brevity
```

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // ...
    self.agent = [SSA getAgentForMediaId: @"someMediaId"];
}
@end
```

Set your MediaID here!

If you need multiple agents for a single *MediaID* you must also provide a `playerId`. The `playerId` has to be unique in the app's scope.

2.3.2 Track User Events with the Agent

Before a video's view time can be tracked, the agent needs to be initialized with the *ContentId* that correspond to the video being tracked.

Whenever the user starts or resumes watching the content, the agent's `notifyPlay` method must be called. Calling this method while the agent is already tracking doesn't have any effect.

When the user stops or pauses watching the content, the agent's `notifyIdle` method must be called. Calling this method while the agent is already paused doesn't have any effect.

The usage of the described methods is shown in the following snippet:

```
#import "VideoPlayerController.h"
#import "SSA.h"

     // some imports omitted for brevity

extern SSA *SSA;
@interface VideoPlayerController () {
    Agent *agent = nil;
     }
    // some properties omitted for brevity
@end

@implementation VideoPlayerController
    // some methods omitted for brevity
- (void)viewDidLoad
{
    [super viewDidLoad];
    // ...
    [self.agent notifyLoadedWithContentId: @"someContentId"
     andCustomParameters: @{@"cp1" :"customParamValue 1"}];
}
- (void)playbackStateChanged
{
```

Set your ContentID here

Set your custom parameters here (max. 4)

```
    switch (self.moviePlayer.playbackState) {
     case MPMoviePlaybackStatePlaying:
         [self.agent notifyPlay]
         break;

     case MPMoviePlaybackStatePaused:
     case MPMoviePlaybackStateInterrupted:
     case MPMoviePlaybackStateSeekingBackward:
     case MPMoviePlaybackStateSeekingForward:
     case MPMoviePlaybackStateStopped:
           [self.agent notifyIdle]
           break;
    }
}
@end
```

2.4 Testing

Once the SSA library has been added to your app, the best way to test or debug the tracking requests is by using a web debugging proxy such as Charles. This way, the measurement can be tested on PC or Laptop.

For successful debugging of the SST integration in your app using Charles, you should use the same WiFi for both your (test-) device and PC/Laptop and set up the PC as Proxy in your mobile device (plus, adding the correct port). After the first contact, Charles should ask if the connection can be trusted and will monitor all of the traffic which can be observed for debugging and testing.

A description of this process can also be found on the Charles website:

http://www.charlesproxy.com/documentation/faqs/using-charles-from-an-iphone/

Alternatively, other web debugging proxies can be used as well, following the same process as described.