UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Mert Celik

Reimplementing the Adcash Software Development Kit in Swift

Master's Thesis (30 ECTS)

Supervisor(s): Marlon Dumas, PhD

Tartu 2017

Reimplementing the Adcash Software Development Kit in Swift

Abstract:

This thesis provides an overview of re-designing and reimplementing of an existing solution, that is used for presenting advertisements in iOS operating system applications. This has been done by reimplementing the existing Objective-C solution in Swift. During reimplementation, the SDK was re-architected in order to take advantage of the features that Swift provides.

Keywords:

Software Development Kit, iOS, Objective-C, Swift, Ad Network
P170 – Informatics:

Adcashi tarkvaraarenduskomplekti ümberstruktureerimine Swift keelde

Lühikokkuvõte:

Käesolevas väitekirjas käsitletakse Objective-C baasil loodud ja iOS operatsioonisüsteemil töötavat reklaamikuvamise rakendust. Magistritöö keskendub rakenduse ümberstruktureerimisele Swift keelde, kasutades viimase uuendusi ning arhitektuurilisi eeliseid.

Võtmesõnad:

Tarkvaraarenduskomplekt, iOS, Objective-C, Swift, reklaamivõrgustik
P170 - Informatics:

Table of Contents

1 Introduction

1.1 Aim of the Thesis

The aim of the work described in this thesis is to build a Software Development Kit (SDK) in Swift 3 for Adcash[1].

Adcash is an advertising network that deals with advertisers (of goods and/or services) and publishers of websites (and any related digital platforms: forums, blogs, etc.). Adcash Advertising Network works as an intermediary between two actors. Adcash recruit publishers and establish contracts with companies (advertisers) that want to display campaigns.

Advertisers invest money to promote a new product or service, expensed in online media buying and website publishers (also called affiliates), earn money every time one advert displayed on his/her website is followed by the user. Thus, Adcash works as a broker between advertiser and publisher. Its role is to ensure performance and efficiency in online advertising.

Adcash operates on both desktop (website) and mobile environments. While websites work with JavaScript tags that are placed within the website, an SDK is needed when publishers want to present advertisements within mobile applications.

Adcash already had Objective-C version of the SDK. Objective-C was the only option to develop applications in iOS operating systems prior to introduction of Swift. After its introduction, Swift gained lot of momentum in terms of adoption rate in market. To be more interoperable and support all languages that Apple offers, we decided to reimplement the SDK in Swift.

The contribution of thesis are an analysis of the architecture and implementation of a pre-existing Adcash SDK for Objective-C, followed by the design and implementation of a new Adcash SDK for Swift. The main outcome of the thesis is the very first mobile ad network SDK in Swift.

1.2 Existing Approaches

Adcash Swift SDK is the very first case where a mobile ad network SDK is ported to Swift. While rewriting entire SDK in Swift is one approach to migrate out of Objective-C, there are two more alternatives; bridging headers and module maps.

Bridging header is a place where we list all the Objective-C class headers we would like to be accessible from Swift. Then in build settings of Swift Compiler, we enter the path of the bridging header. This allows publishers to use Objective-C code in their Swift applications. MoPub[2] and Revmob[3] is using this approach in their mobile SDK's.

Module maps on the other hand, is the description between modules and headers. It describes which header maps on which structure of the module. Compiler uses module maps

[1] https://www.adcash.com/
[2] https://www.mopub.com/
[3] https://www.revmobmobileadnetwork.com/

to automatically generate binary representation of the modules. AdMob[4] and InMobi[5] are an example of ad networks that are using module maps in their SDK's.

While the use of bridging headers or module maps do entail any significant performance overhead, they do not allow us to take advantage of the more sophisticated features of the Swift programming language.

At Adcash, we saw an opportunity to become the first provider of a mobile ad network SDK implemented in Swift and accordingly, decided to invest time building the SDK, in favor of Swift's future and the Adcash developer's (Author) professional development.

1.3 Terminology

The following terms are used throughout the thesis.

Software Development Kit (SDK) is a group of code that is used for developing applications for a specific device or an operating system.

Advertiser is the owner of the business who wants to advertise their product in ad network to get targeted with quality traffic.

Publisher is the owner of the mobile application who can use SDK in their application to make money. In this work, they will be also referred as application developers.

MRAID is an abbreviation for Mobile Rich Media Ad Interface Definitions. It is a standard set by Interactive Advertising Bureau[6].

Impressions are referring to the number of time an ad shown to users.

Zone ID is automatically created ad space number in server.

Banner is a rectangular image or text ads that occupy a spot within application's layout. An example can be seen in Figure 1 below.

Figure 1: Example banner ad.

[4] https://www.google.com/admob/
[5] http://www.inmobi.com/
[6] https://www.iab.com/

Interstitial is a full screen ad that covers the interface of their host application. An interstitial example can be seen in Figure 2 below.

Figure 2: Example interstitial ad.

1.4 Outline

The thesis is divided into five chapters and an appendix.

This is the first chapter that introduces the aim of the thesis.

The second chapter gives background information about Adcash Objective-C SDK and Swift.

The third chapter provides an overview of Adcash Objective-C SDK architecture, its implementation and possible improvements.

The fourth chapter provides and overview of Adcash Swift SDK architecture and its implementation.

The fifth and last chapter provides the conclusion and future work of the Adcash Swift SDK.

The appendix is a XML file containing VAST template for a video ad.

2 Background

In this chapter we will give a short overview of Adcash SDK, Swift and possibilities that comes with it.

2.1 Adcash Software Development Kit

Advertisers use advertising (ad) networks to serve their advertisements and ad networks deliver ads to publishers' applications from an ad server. In order to make request and to receive an ad from the ad server, SDK of the ad network needs to be added to publisher's application. Using the code that comes with the SDK, the publisher's application sends the data that are obtained from the user's phone and receives an advertisement based on the parameters that are sent in the request. This usage results in higher quality ad placement since the selection of the ad is more relevant to the user of the device.

Adcash Objective-C SDK helps mobile application developers to monetize their applications and maximize their revenues. It allows publishers to put ads in their applications, using native User Interface (UI) components.

2.2 Advantages of using Swift

Objective-C is a an Object Oriented Programming (OOP) language that adds Smalltalk[7]-style messaging to the C programming language. Objective-C was the main language in Apple's operating systems and Application Programming Interfaces (API) until the introduction of Swift in 2014. Swift was designed to take advantage of the perceived advantages of Objective-C, particularly with regards to its objective-oriented model, but at the same time to remove some of the idiosyncrasies inherited from the C language. The design of Swift was driven by the slogan that Swift should be "Objective-C without the C." [1]

Swift still has access to most of the capabilities of Objective-C but it introduces many advantages that are not available in Objective-C. Below, you will find brief explanations of some advantages that are used in this work:

Swift is easier to maintain. Objective-C has two distinct compilation units for each file; header file with .h extension and implementation file with .m extension. The implementation file has the body of the functions, but header file is where they are declared and their access control properties set. This requires developers to maintain two code files in order to improve the build time and efficiency of the application. In Swift, Xcode[8] and the Low Level Virtual Machine[9] (LLVM) can figure it out dependencies and perform incremental builds automatically. As a result, the repetitive task of separating header and implementation files are combined in a single code file in Swift with .swift extension.

Swift is safer. When you call a pointer value that is uninitialized in Objective-C, the operation is skipped altogether. While it is good that it doesn't crash, it leads to unpredictable behavior. Swift introduces a value type called optional. It is a type that can hold either a value or no value. To provide a predictable behavior, Swift triggers runtime

[7] https://en.wikipedia.org/wiki/Smalltalk
[8] https://developer.apple.com/xcode/
[9] http://llvm.org/

error if a nil optional variable is used. This creates a short feedback loop and eases the bug-fixing progress because it forces the developer to fix the issue right away.

Swift has better performance compared to Objective-C. While Objective-C is a superset of C language, Swift is a language built from the ground, up for the LLVM compiler, leaving the limitations of C. It also has better memory management compared to Objective-C. Object links in Objective-C uses random-access memory but when Apple implemented Swift, they left the OOP and switched to structures. This resulted moving from reference type to value type and led to more efficient memory usage in Swift. Since Swift is static language and all types are known before runtime, compiler can optimize the code without any problem. On the other hand, Objective-C cannot be optimized as effectively as Swift because of its dynamic typing.

Swift is a multi-paradigm language. Swift can be used to write object-oriented code or pure functional code using immutable values, or even imperative C-like code using pointer arithmetic.

3 Objective-C SDK Architecture

This chapter gives an overview of Adcash Objective-C SDK, explains its components implementation and discusses possible improvements.

Objective-C is known as an OOP language. Adcash developers, who started the development of Adcash Objective-C SDK, naturally followed an object oriented design concept while building the SDK.

If we look into the architecture, we can see that SDK is designed around three main components:

- Networking Component
- Nexage Integration SourceKit for Mobile Rich Media Ad Interface Definitions
- Custom Components

These components will be explained in detail, later in this chapter.

The initial version of the Adcash Objective-C SDK supported banner and interstitials. The general flow of the SDK for both ad formats for initializing and displaying the ad shown in Figure 3:

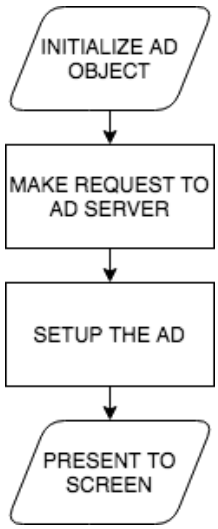

Figure 3. Flow for setting up and presenting an ad.

Publisher starts the progress by creating and initializing an ad object in their application with providing their zone ID as a parameter. Within the initialization of ad object, SDK runs necessary code to collect parameters from user's device and uses these parameters to make a request to ad server, using networking component. Later on, SDK sets up the UIWebView[10] with frame size based on the ad type and feeds the received HTML into UIWebView. Again with the publisher's call in code, SDK attaches the UIWebView to main window to display the ad.

Since the flow is same for both ad formats, there are repetitive steps such as collecting user parameters with every ad request. All repetitive code can be moved into a singleton

[10] https://developer.apple.com/documentation/uikit/uiwebview

SDK class, that would reduce the amount of code clones and increase the efficiency of the code.

3.1 Networking Component

The developers who started developing Adcash Objective-C SDK used an open source library called AFNetworking [2] to design the client-side API on mobile devices.

Since there is no internal documentation about why AFNetworking is chosen, we are presenting our own analyse between few popular networking libraries on GitHub[11] to see and discuss if the decision made is successful by growth of libraries achieved until July, 2017.

Table 1: Popular open-source iOS networking libraries from GitHub

Name	Initial release	Star	Fork	Contributors	Issues (open/closed)
AFNetworking	Jun 4, 2011	29,625	9,293	274	303 / 2,310
ASIHTTPRequest [3]	Jul 26, 2008	5,790	1,503	49	116 / 167
CocoaAsyncSocket [4]	Apr 2, 2012	8,907	2,407	40	237 / 202
FSNetworking [5]	Jun 29, 2012	402	66	4	6 / 9
RestKit [6]	Mar 3, 2011	10,100	2,252	163	323 / 1,508

3.1.1 AFNetworking

AFNetworking is an open-source HTTP library for iOS and Mac OS X. It's built on top of Foundation network classes[12], using NSOperation[13] for asynchronous execution and blocks[14] for convenience. [7] This is the library chosen by developers who started Adcash Objective-C SDK. It has clean and understandable syntax and very easy to use as seen in Snippet 1.

```
[self GET:@"" parameters:parameters
   success:^(AFHTTPRequestOperation *operation, id responseObject) {
       //use response
   } failure:^(AFHTTPRequestOperation *operation, NSError *error) {
       //user error
   }
];
```

Snippet 1: GET request with AFNetworking.

In 6 years and 1 month, AFNetworking growth their community with 274 contributors while managing issues open / closed ratio by 0,14. Also library has 29,625 stars that means two distinct actions; creating a bookmark for easy access or showing appreciation to the repository maintainer for their work[15].

[11] https://github.com/
[12] https://developer.apple.com/documentation/foundation/url_loading_system
[13] https://developer.apple.com/documentation/foundation/nsoperation?language=objc
[14] https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/WorkingwithBlocks/WorkingwithBlocks.html
[15] https://help.github.com/articles/about-stars/

3.1.2 ASIHTTPRequest

ASIHTTPRequest is another HTTP library that makes communication with servers easier. When we check its implementation of our needs as presented in Snippet 2. It is worth noting that blocks and asynchronous request is not the default implementation of the library and it can be achieved by developer with snippet below:

```
NSURL *url = [NSURL URLWithString:@"API_URL"];
__block ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setCompletionBlock:^{
   //use response
}];

[request setFailedBlock:^{
   //use error
}];

[request startAsynchronous];
```

Snippet 2: GET request with ASIHTTPRequest.

The owner of the ASIHTTPRequest stopped maintaining the library from May, 2011. Therefore, we assumed that all the values in Table 1 collected until that date, which means 2 years and 9 months. In this time frame, library managed to grew its community to 49 contributors with keeping issue ratio of 0,69.

3.1.3 CocoaAsyncSocket

CocoaAsyncSocket another popular library that provides a good and easy API for working with socket connections. It doesn't have RESTful-based services implemented by default which will require lot of boilerplate code from our side. On the other hand, the Adcash Ad Server deals big amount of connections in real time, it would be very complex to keep big amount of socket connections open. Therefore, this library is not a good fit for our needs.

3.1.4 FSNetworking

FSNetworking is a library developed by Foursquare labs as a replacement for ASIHTTPRequest in their application. It is a small library in a single class, supports asynchronous, block based GET and POST requests. Since GET request is the only service we need, this library would be more good fit compared to AFNetworking because it doesn't have the boilerplate code for the features we don't use.

```
FSConnection *connection = [FSConnection withURL: url
method: FSNRequestMethodGET
headers: headers
parameters: parameters
parseBlock:^id(FSNConnection *c, NSError *error){
   return [c.responseData dictionaryFromJSONWithError: error];
}completionBlock:^(FSNConnection *c){
   //use result
}];
[connection start];
```

Snippet 3: GET request with FSNetworking.

FSNetworking received its last commit in Feb, 2013 and haven't got maintained since. It received less stars, forks and has less contributors compared to other libraries. It is because the library designed for GET and POST requests only. It has issue ratio of 0,67.

3.1.5 RestKit

RestKit is a library that implements RESTful web service clients in iOS and Mac OS X. It is built on top of AFNetworking. Even though it provides easy and powerful response to object mapping, being built on top of AFNetworking and having way more features than we need means more boilerplate code and growth on the size of SDK.

```
NSURLRequest *request = [NSURLRequest requestWithURL:[NSURL
URLWithString:@"API_URL"]];

RKObjectRequestOperation *operation = [[RKObjectOperation alloc]
initWithRequest:request];

[operation setCompletionBlockWithSuccess:^(RKObjectRequestOperation *operation,
RKMappingResult *result) {
   //use response
} failure:^(RKObjectRequestOperation *operation, NSError *error){
   //use error
}];

[operation start];
```

Snippet 4: GET request with RestKit.

In 6 years and 4 months, RestKit grew its community to 163 contributors while maintaining issue ratio of 0,21. Library received a star count of 10,100.

3.1.6 Synthesis

Since we removed CocoaAsyncSocket from options, we concluded our analyze with four remaining libraries.

From implementation point of view, AFNetworking is the most practical to use since others need an explicit start call for starting the request, while whole request can be made with one call in AFNetworking.

On the maturity side, when issue ratios are compared as seen in Figure 4, which means the number of open issues contrary to 1 closed issue. If we use the ratio as the responsiveness to bugs of the community, AFNetworking is again, the winner.

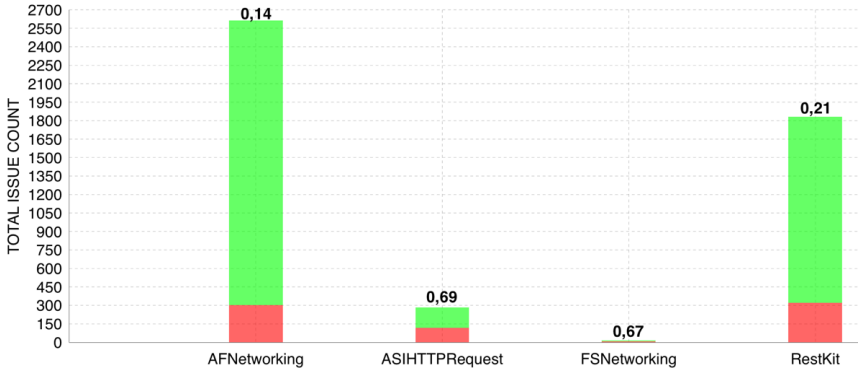

Figure 4: Issue ratio of the Objective-C networking libraries.

3.2 Nexage Integration SourceKit for MRAID

Developers who started the development of Adcash Objective-C, used an open source library called "Nexage Integration SourceKit for Mobile Rich Media Ad Interface Definitions (MRAID)" [8]. It is written in Objective-C and works with both iPhone and iPad.

MRAID is a standard set by Interactive Advertising Bureau (IAB) to help advertisers deal with the issues caused by variety of devices running on different operating systems. IAB defined a common API for ad developers. For which, the API is used for mobile rich media ads so they can run in mobile apps. The API is a standardized set of commands designed to work with HTML5 and JavaScript. So, when ad developers create rich media ads, they can use the command to communicate with mobile apps to perform actions they want. For instance, actions like expand, resize, get information, access native apps like video player, etc.

The library made by Nexage Inc., which later on acquired by Millennial Media [9], implements the IAB MRAID 2.0 specifications[16]. This solution was lagging behind the market because it did not have the standards that were introduced with Video Ad Serving Template (VAST) standard by IAB. Before this standard they did not support video ads at all.

The IAB's description of VAST is "a standard XML-based ad response for in-stream video as well as an XML Schema Definition for developers." [10]. VAST contains the information about the ad that is given to the video player, such as; links of videos in different quality and bitrates, duration, tracking URL's and so on. When the video player adopts the VAST schema, player will be able to read and display video ads from other ad servers as well.

It is important to point out that VAST and MRAID implementations will be handled as separate component in the Adcash Swift SDK. Since the MRAID implementations other than video ads are still same, all framework will be rewritten in Swift with no changes in behaviour. In order to make Adcash Swift SDK compatible with VAST files, an XML parser component for reading the XML and video player capable of working with VAST file will be developed.

3.3 Custom Components

This section provides an information about the custom classes of Adcash Objective-C SDK, discusses their current implementation and proposes improvements.

To make the explanation of the custom components better, we can group them into subcomponents as seen in Table 2.

[16] http://www.iab.com/wp-content/uploads/2015/08/IAB_MRAID_v2_FINAL.pdf

Table 2: Custom components grouped by their purpose.

Ad Type Specific	Networking	Internal
• ADCBannerView • ADCBannerViewSizeProvider • ADCBannerVisibilityObserver • ADCTimerManager • ADCBannerReloader • ADCInterstitial	• ADCAPIClient	• ADCSDK • ADCRequest • ADCError

3.3.1 Internal

This section provides an information about the helper classes of Adcash Objective-C SDK, discusses their current implementation and proposes improvements.

3.3.1.1 ADCSDK

ADCSDK is a class responsible for initializing itself as a singleton object. It stores the release version of the Adcash Objective-C SDK as a string.

Even though it is right way to have a singleton SDK object to prevent fraud cases such as having multiple SDK instances working in parallel, this class is poorly designed and doesn't do anything else than initializing the singleton and returning version string. It can be improved by storing generic values and calculations that will be needed as long as publisher's application is in use.

3.3.1.2 ADCRequest

ADCRequest is the class that gathers information about user's device, such as:

- Advertising Identifier
- Screen Size
- Time Zone
- Device Type
- Device Language
- Application Name
- Location (if allowed)
- ...

This class, is also responsible for returning necessary parameters to send with request to ad server. Implementation of the class is pretty straight forward but since it's not a singleton object it initializes itself and calculates every value for every single request.

Only 3 values out of 18 has the possibility to get updated within single application session. Current usage causing them to be updated with every request. It means that for every ad request, this class is initialized and 18 functions to get these values are called.

3.3.1.3 ADCError

ADCError is a struct that contains representations of errors such as,

- ADCErrorNoFill (When there is no ad returned.)
- ADCErrorInternalError (When there is an error caused by SDK.)
- ADCErrorInvalidRequest (When invalid request made to ad server.)

3.3.2 Networking

This section provides an overview of networking classes of Adcash Objective-C SDK, built by using open-source library AFNetworking.

3.3.2.1 ADCAPIClient

ADCAPIClient class is responsible with all network requests made from SDK to ad server. It is also designed as a singleton object to prevent the situation of one SDK having multiple network connections at the same time. It takes parameters gathered by ADCRequest and sends it to ad server to get an ad. It also checks if the device is connected to internet. In case of an error happens, it fails with results that are mentioned in ADCError.

Adcash Objective-C SDK is using AFNetworking for making requests to Adcash Ad Server. Since SDK is only using GET requests, ADCAPIClient is very lightweight. It has only three functions to use and their usage is presented in Snippet 5:

- **sharedClient** is for to create and initialize singleton ADCAPIClient object and start monitoring for internet connection,
- **fetchWithRequest** for making request to Adcash Ad Server, with collected parameters added.
- **ADCErrorWithResponse** is to get corresponding error code for the failed response.

```
[[ADCAPIClient sharedClient] fetchWithRequest: request zoneID: self.zoneID
success:^(AFHTTPRequestOperation * operation, id responseObject) {
   // Use response
} failure:^(AFHTTPRequestOperation *operation, NSError *error) {
  // Present error
}];
```

Snippet 5: GET request implementation from Adcash Objective-C SDK.

3.3.3 Ad Type Specific

Adcash Objective-C SDK supported banner and interstitial ad types. Banner ads stand in screen while users are interacting with the app and can refresh automatically after a certain period of time. While it has mobility to be placed any point within application's layout, in mobile products of Adcash, we only support them to be at the bottom of the screen and call them "footers".

In case of showing interstitial, user has the choice to either tap on the ad and continue to its destination URL, or close it and return to the application.

Both ADCBannerView and ADCInterstitial classes are wrapper classes [11] around SKMRAIDView, that creates the UIWebView to display the ad. Even though they are two different ad types, from technical point of view they are just UIWebViews that presents HTML on different frame sizes.

3.3.3.1 ADCBannerView

When the publisher wants to show a banner on UIViewController in their application, they have to do it with following the steps below:

First, they have to import the library into their UIViewController as shown in Snippet 6. This allows developers to access the public classes in the Adcash Objective-C SDK.

```
#import "AdcashSDK.h"
```

Snippet 6: Importing Adcash Objective-C SDK.

Second, as an optional step, they can make their UIViewController class conform to ADCBannerViewDelegate as shown in Snippet 7, if they want to receive updates about the ad's delegates[17].

```
@interface SomeViewController : UIViewController <ADCBannerViewDelegate>
@end
```

Snippet 7: Conforming ADCBannerView delegate.

And as the last mandatory step, they have to implement code shown in Snippet 8, into their viewDidLoad: method of UIViewController[18] subclass.

```
//Initialize the ADCBannerView
ADCBannerView *bannerView = [[ADCBannerView alloc]
                 initWithZoneID:@"your_zone_id"
              onViewController:self];
// Do not translate autoresizing mask into constraints
bannerView.translatesAutoresizingMaskIntoConstraints = NO;

// Add your banner as a subview to your view
[self.view addSubview:bannerView];

// Add constraints to the banner
// Set the banner take the width of it's parent view
NSDictionary *views = NSDictionaryOfVariableBindings(bannerView);
[self.view addConstraints:[NSLayoutConstraint
constraintsWithVisualFormat:@"H:|[bannerView]|"
                                    options:0
                                        metrics:nil
                                          views:views]];
// Set the banner to stick to the bottom of it's parent
[self.view addConstraints:[NSLayoutConstraint
constraintsWithVisualFormat:@"V:[bannerView]|"
                                        options:0
                                        metrics:nil
                                          views:views]];
// Set the banner's delegate to be your view controller
bannerView.delegate = self; //optional

[bannerView load];
```

Snippet 8: Initialization and loading of ADCBannerView.

[17] https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaEncyclopedia/DelegatesandDataSources/DelegatesandDataSources.html

[18] https://developer.apple.com/documentation/uikit/uiviewcontroller

If publishers implemented the optional second step, they have access to the delegate methods shown in Snippet 9.

```
-(void) bannerViewDidReceiveAd:(ADCBannerView *)bannerView;
-(void) bannerView:(ADCBannerView *)bannerView didFailToReceiveAdWithError:(NSError
*)error;
-(void) bannerViewWillPresentScreen:(ADCBannerView *)bannerView;
-(void) bannerViewWillLeaveApplication:(ADCBannerView *)bannerView;
-(void) bannerViewWillDismissScreen:(ADCBannerView *)bannerView;
```

Snippet 9: ADCBannerViewDelegate methods.

To understand the logic of how above code works, we can check explanations below:

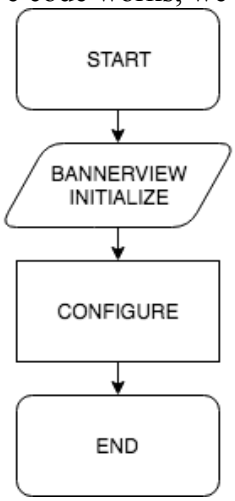

Figure 5: Banner initialization flow.

With the initialization of the banner class as presented in Figure 5 , SDK initializes the ADCBannerView class and save the zone ID variable to use it in request to ad server. In this call, UIViewController is referenced to calculate the frame size and attach the banner later on.

After initialization, SDK follows the flow presented in Figure 5 with the call of load function.

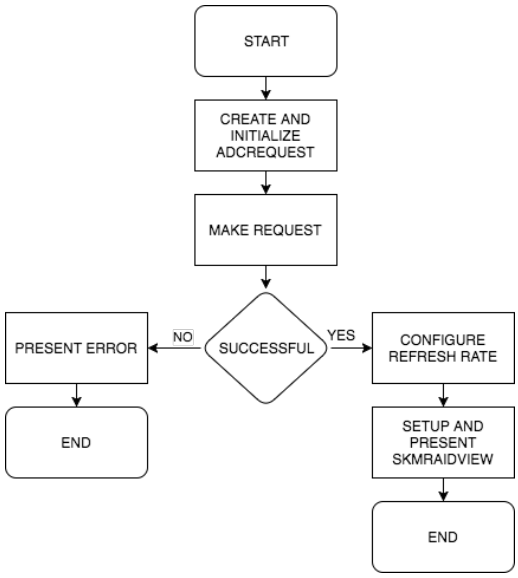

Figure 5: Banner load flow.

With the call of load function:

- SDK initializes and stores ADCRequest class with parameters collected from user's device.
- Then using those parameters, SDK makes a request to ad server.
- In case of failure, SDK sends bannerViewDidFailToReceiveAd: delegate and logs the error to the debugger of Xcode.
- In case of success, SDK sends bannerViewDidReceiveAd: delegate and reads refresh rate value from response's header to start the ADCBannerReloader class that works together with ADCTimerManager and ADCBannerVisibilityObserver.
- As the last step, SDK feeds the received HTML file into the SKMRAIDView and attaches it to screen.

ADCBannerView class has more complex structure when it's compared to ADCInterstitial class. Since banners are staying on screen without closing option, they have more requirements such as reloading and visibility. These requirements are satisfied with ADCBannerReloader, ADCTimerManager and ADCBannerVisibilityObserver classes. Since these classes are only related to banner and require no further development, they can be moved to banner class extension. Moving them into extension will isolate the stable code and make it easier to maintain.

3.3.3.2 ADCInterstitial

When the publisher wants to show an interstitial in their application, they have to do it with following the steps below:

First, they have to import the library into their UIViewController as shown in Snippet 10. This allows developers to access the public classes in the Adcash Objective-C SDK.

```
#import "AdcashSDK.h"
```

Snippet 10: Importing the Adcash Objective-C SDK.

Second, they must declare an interstitial property as seen in Snippet 11. As an optional step, they can make their UIViewController class conform to ADCInterstitialDelegate, if they want to receive updates about the ad's state.

```
@interface SomeViewController : UIViewController <ADCInterstitialDelegate>
@property (nonatomic,strong) ADCInterstitial *interstitial;
@end
```

Snippet 11: ADCInterstitial instance and conforming ADCInterstitialDelegate

And as the last mandatory step, they have to implement the code shown in Snippet 12, into viewDidLoad: method of their UIViewController subclass.

```
// Assign an ADCInterstitial instance to your property.
self.interstitial = [[ADCInterstitial alloc] initWithZoneID:@"your_zone_id"];
// Make your view controller a delegate to the interstitial.
self.interstitial.delegate = self; // Optional
// Load the interstitial.
[self.interstitial load];
```

Snippet 12: Initialization and loading of ADCInterstitial.

When the interstitial loaded successfully, publisher can catch the status update (if delegate is conformed) and present it to the screen with the code shown in Snippet 13.

```
- (void)interstitialDidReceiveAd:(ADCInterstitial *)interstitial{
    [interstitial presentFromRootViewController:self];
 }
```

Snippet 13: Presenting the interstitial to screen.

If publisher implemented the optional step, they will have access to functions presented in Snippet 14.

```
- (void) interstitialDidReceiveAd:(ADCInterstitial *)interstitial;
- (void) interstitial:(ADCInterstitial *)interstitial
didFailToReceiveAdWithError:(NSError *)error;
- (void) interstitialWillPresentScreen:(ADCInterstitial *)interstitial;
- (void) interstitialWillDismissScreen:(ADCInterstitial *)interstitial;
- (void) interstitialWillLeaveApplication:(ADCInterstitial *)interstitial;
```

Snippet 14: ADCInterstitialDelegate methods.

To understand how interstitial works under the hood, we can refer the explanations below:

As shown in Snippet 12, SDK initializes ADCInterstitial object and saves the zone ID for using with request. After the call of load function, SDK follows the flow presented in Figure 6.

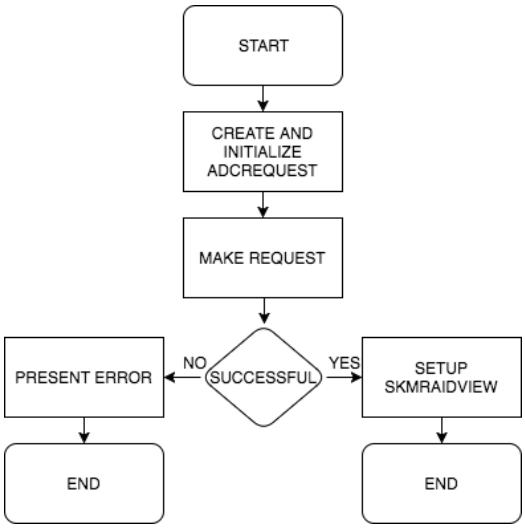

Figure 6: Interstitial load flow.

With the call of load function:

- SDK initializes and stores ADCRequest class with parameters collected from user's device.
- Then using those parameters, SDK makes a request to ad server.
- In case of failure, SDK sends interstitialDidFailToReceiveAd: delegate and logs the error to debugger of XCode.
- In case of success, SDK feeds the received HTML file into the SKMRAIDView.

Since it's not automatically displayed on screen like banners, publisher needs to call presentFromRootViewController function to present it on screen as seen in Figure 7.

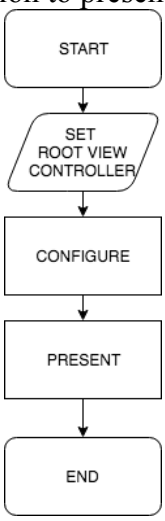

Figure 7: Presenting interstitial flow.

With the presentFromRootViewController function,

- SDK sets the root view controller to attach interstitial.
- Then uses the root view controller's window frame to make SKMRAIDView full screen.
- And attaches the SKMRAIDView to the root view controller.

4 Swift SDK Architecture

This chapter gives an overview of Adcash Swift SDK, explains its components implementation and presents improvements made.

Adcash Swift SDK followed same component structure as Adcash Objective-C SDK. It has three main components as networking, internal and ad specific components.

4.1 Networking

The networking component of Adcash Swift SDK is also built by using an open-source framework. In order to select an open-source framework to develop this component, we focused on two criteria: maturity and being written natively in Swift 3.

The reason we decided to go for libraries written in Swift 3, is speed. When we have Objective-C and Swift compiling in the same program, the speed is defined by compiler technology. Swift uses LLVM compiler to optimize the code. For example, by using optionals, Swift ensures that certain pointers can never be nil, so the compiler can omit nil checks. On the other hand, since structs are first class citizens in Swift, optimizer assumes that constants never change so it can cache previously-fetched values and focus on performing other optimizations. This ends up Swift having faster execution and more optimized code.

As the other criteria, for maturity, we considered the level of community involvement and activity in project, regarding to development. When we listed top 5 frameworks written in Swift 3 on GitHub, we have the table below:

Table 3: Popular open source networking libraries.

Name	Initial release	Star	Fork	Contributors	Issues (open/closed)
Alamofire [12]	Sep 26, 2014	24,588	4,267	142	9 / 1,657
Just [13]	May 9, 2015	1,022	92	12	14 / 39
Networking [14]	Feb 22, 2015	1,045	78	9	14 / 56
Pitaya [15]	May 15, 2015	847	93	3	2 / 17
Siesta [16]	Aug 28, 2015	1,342	101	16	10 / 115

4.1.1 Alamofire

Since Alamofire is developed by the same community that is working on AFNetworking, it has the familiar API design with some improvements such as validations, when compared the version used in Adcash Objective-C SDK. Validations help us tackle issues like receiving different content than we expected. Receiving banner, while requested for video can be one of the examples.

With using validations, we can set rules for ad classes, such as; “application/xml” type for video, “text/html” for banners and interstitials and “application/json” for native ads. This provides security for crashing, in case ad server fails to response with the correct ad type. Snippet 15 shows the implementation Alamofire for our use, making GET request.

```
Alamofire.request(kAdcashAPIBaseURL, parameters: params)
        .validate(contentType: ["text/html"])
        .responseString { (response) in
           switch response.result {
           case .success(let value):
               // Use value
           case .failure(let error):
               print(error.description)
           }
         }
```

Snippet 15: GET request with Alamofire.

In 2 years and 9 months, Alamofire grew its community to 142 contributors while maintaining issue ratio of 0,005. Even though having great community from AFNetworking helped this library to grew this big, it has 4,267 forks which means a lot of people copied the repository and experimenting with it. The library also has 24,588 star, which proves that the library is very reliable and adopted by lot of developers.

4.1.2 Just

Just's API, as stated in their documentation[19], is heavily influenced by python requests. As seen in Snippet 16, it has very minimalistic API design for basic HTTP requests. Just is very easy to use, but there is no default implementation for validations for content type. Therefore, it needs to be implemented by users, which leads to more boilerplate code.

```
var r = Just.get(kAdcashAPIBaseURL, params: params)
If (r.ok){ // status code is not 4xx or 5xx
   let response = r.text
   // Use response
} else {
   print(r.reason)
}
```

Snippet 16: GET request with Just.

Just grew its community to 12 contributors in 2 years and 2 months while maintaining 0,36 issue ratio. The library collected 1,022 stars and 92 forks.

4.1.3 Networking

Networking is another library that implemented with a very similar API design with Alamofire but needs manual implementations for validations, like it is in Just. Example of GET request can be seen in Snippet 17.

```
let networking = Networking(baseURL: kAdcashBaseAPIURL)
networking.get("/get", parameters: params) { result in
   switch result {
   case .success(let response):
       // Use response
   case .failure(let error):
       // Present error
   }
}
```

Snippet 17: GET request with Networking.

[19] https://github.com/JustHTTP/Just/blob/master/README.md

In 2 years and 5 months, Networking worked with 9 contributors and maintained issue ratio of 0,25. Library collected 1,045 stars and 78 forks.

4.1.4 Pitaya

Pitaya is yet another library that is inspired[20] by Alamofire and Just. Even though it's inspired by Alamofire, same situation with validation persists. GET request implementation by Pitaya can be seen in Snippet 18.

```
Pita.build(HTTPMethod: .GET url: kAdcashAPIBaseURL)
    .addParams(params)
    .onNetworkError({ (error) → Void in
        // Present error
    })
    .responseString { (string, response) → Void in
        // Use response
    }
```

Snippet 18: GET request with Pitaya

On the other hand, it has advanced options implemented very easy to use, such as SSL pinning and uploading files.

In 2 years and 2 months, Pitaya collected 847 stars and 93 forks. 2 contributors of Pitaya maintained issue ratio of 0,18.

4.1.5 Siesta

Siesta has completely different design compared to others, by providing client-side cache of observable models for RESTful resources. Since it has a different implementation as seen on Snippet 19, it requires change in implementations of ad class designs. Therefore, it doesn't fit our needs.

```
let MyAPI = Service(baseURL: kAdcashAPIBaseURL)
MyAPI.resource.addObserver(self)

func resourceChanged(_ resource: Resource, event: ResourceEvent) {
    // Use response
}
```

Snippet 19: GET request with Siesta.

In a timeframe of 1 year 11 months, 16 contributors maintained an issue ratio of 0,09. The library collected 1,342 stars and 101 forks. Even though it looks more mature than Pitaya, Networking and Just by means of stars and forks, implementation difference requires big changes in the SDK design. Therefore, Siesta is not a good fit for our needs.

4.1.6 Synthesis

After removing Siesta from options, we have concluded our selection between remaining frameworks.

[20] https://github.com/johnlui/Pitaya/blob/swift3/README.md

As seen from the examples in above, none of the libraries has significant value over each other by means of implementation. Even though Just has very minimalistic design, it is not mature enough to be considered as a winner. This, again, leads us to decide on level of responsiveness in community regarding fixing issues as presented in Figure 8.

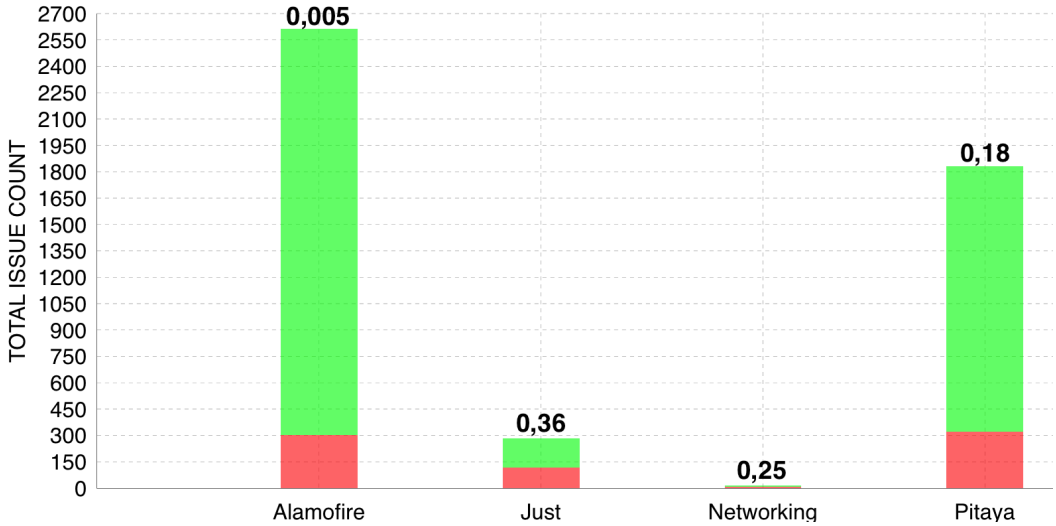

Figure 8: Issue ratio of the Swift networking libraries.

As seen above, Alamofire outperformed other communities by having 0,005 issue ratio. Since Alamofire has familiar API design from AFNetworking and has very responsive community, we have selected Alamofire as the networking component of Adcash Swift SDK.

4.2 Internal

This section provides an information about the helper classes of Adcash Swift SDK and discusses how they differ from their Adcash Objective-C SDK correspondents.

4.2.1 AdcashConstants

AdcashConstants is a singleton class that holds both static and dynamic variables about SDK and device, such as:

- Adcash API URL
- AdcashBrowser tool bar height
- Current orientation of the device
- Calculated AdcashBanner size regarding to the screen size
- Device type (iPhone or iPad)
- Calculated size values of all visual objects (UIButton, UILabel) that are used in SDK
- Error codes.

There are few motivations behind this class:

First of all, Swift offers great possibilities with access control. Instead of having all visuals stored and re-calculated for every format in their classes, we now have access the screen size information already calculated regarding to the screen size, ready to be read.

Since these values can't be changed afterwards, we can have these calculated and stored as static values.

For example, to get the banner height we check the device type with the line presented in Snippet 20 and get the corresponding size as show in Snippet 21.

```
let isIpad: Bool = (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiom.pad ? true:
false)
```

Snippet 20: Device type retrieval.

```
public var AdcashBannerSize: String {
   If isIpad {
       return "90"
   }
   return "50"
}
```

Snippet 21: AdcashBannerSize variable.

Values presented in AdcashBannerSize are hard coded for protecting the ad's visual quality with respecting the scale of the ad.

As an another example, we can access to size values of visual elements like the struct presented in Snippet 22.

```
struct AdcashVideoControlsPad {
   let preRollMessageButtonWidth: CGFloat = 300.0 // For the "WATCH NOW" button.
   let preRollMessageButtonHeight: CGFloat = 80.0
   let rollMessagesFontSize: CGFloat = 32.0 // Custom texts like "Watch to earn 10
coins"
   let buttonTitleSize: CGFloat = 30.0
   let closeRegionSize: CGFloat = 65.0 // closing area for pre and post roll messages
}
```

Snippet 22: Video controls struct.

As a second motivation, this approach makes it easier to address user interface problems and fix them in single component. On the other hand, now it is more convenient to try new ad sizes for current ad formats or experiment on new formats.

And as a final motivation, having all internal values, such as error codes and Adcash Base API URL that are shown in Snippet 23 helps us easily switch between development and production level API's of Adcash.

```
enum kAdcashErrorCode: Error {
   case noFill // No ads returned.
   case internailError // For errors caused within SDK
   case invalidRequest  // If request doesn't have necessary parameters
}

enum kAdcashAPIBaseURL: String {
   case production = "production URL"
   case development = "development URL"
}
```

Snippet 23: kAdcashErrorCode and kAdcashAPIBaseURL structs.

4.2.2 AdcashRequest

The ADCRequest class which is used for gathering parameters from device in Adcash Objective-C SDK had the problem of initializing with every ad request. This problem of having ADCRequest class initialize for every ad request is removed with the AdcashConstants. AdcashConstants initializes the AdcashRequest within its singleton implementation and populates the parameters dictionary and returns it with another method both shown in Snippet 24.

```
parameters = ["did": deviceType,
              "o": interfaceOrientation
              "appn": appName
              "geo": deviceLocation
              .. //and so on
]

func dataParameters() -> Dictionary<String, String> {
   return self.parameters
}
```

Snippet 24: Gathered parameters and its return function.

Every time a request made dataParameters() function fetches the data from singleton class and populates the request. This approach solves the problem of wasting resources by re-calculating values for every ad request.

4.2.3 AdcashView

Since AdcashView is replacing the SKMRAIDView and used the display both interstitials and banners, it would be very accurate to call it the most important class of Adcash Swift SDK. By replacing SKMRAID, we remove the boilerplate code that is not used such as logging and we introduce the object-value layer separation [17], that is proposed by Andy Matuschak, with StateValue struct.

When we compare the implementations of Objective-C and Swift SDK in Snippet 25 and Snippet 26 it is easier to see, the readability of the code is much better in new implementation.

```
-(id)initWithFrame:(CGRect)frame
             html:(NSString*)htmlString
             baseURL:(NSString*)bsURL
             delegate:(id<SKMRAIDViewDelegate>)delegate
             rootViewController:(UIViewController*)rootViewController {...}
```

Snippet 25: SKMRAID initialization in Adcash Objective-C SDK.

```
init(type: AdType,
    state: StateValue,
    delegate: AdcashProtocol,
    rootVC: UIViewController)
```

Snippet 26: AdcashView initialization in Adcash Swift SDK.

With the help of AdType enumeration as seen in Snippet 27, we retrieve relevant size information from AdcashConstants to set up the size of AdcashView.

```
enum AdType {
   case banner
   case interstitial
}
```

Snippet 27: AdType enumeration.

On the value layer as presented in Snippet 28, having two values stored helps us handling responses both, sent as HTML file or as a base URL.

```
struct StateValue {
   let html: String
   let baseURL: URL
}
```

Snippet 28: StateValue struct.

4.2.4 AdcashProtocol

Adcash Objective-C SDK has separate protocol implementations for each ad class. Even though their functionality is exactly the same, they are implemented in banner and interstitial classes with different names. If we compare the implementations that are presented in Snippet 29 and Snippet 30:

```
@protocol ADCBannerViewDelegate <NSObject>
@optional
-(void) bannerViewDidReceiveAd:(ADCBannerView *)bannerView;
-(void) bannerView:(ADCBannerView *)bannerView didFailToReceiveAd:(NSError *)error;
-(void) bannerViewWillPresentScreen:(ADCBannerView *)bannerView;
-(void) bannerViewWillLeaveApplication:(ADCBannerView *)bannerView;
-(void) bannerViewWillDismissScreen:(ADCBannerView *)bannerView;
@end
```

Snippet 29: ADCBannerViewDelegate methods.

```
@protocol ADCInterstitialDelegate <NSObject>
@optional
-(void) interstitialDidReceiveAd:(ADCInterstitial *)interstitial;
-(void) interstitial:(ADCInterstitial *)interstitial didFailToReceiveAd:(NSError
*)error;
-(void) interstitialWillPresentScreen:(ADCInterstitial *)interstitial;
-(void) interstitialWillLeaveApplication:(ADCInterstititat *)interstitial;
-(void) interstitialWillDismissScreen:(ADCInterstitial *)interstitial;
@end
```

Snippet 30: ADCInterstitialDelegate methods.

It is easy to see that they exist for same purpose and can be unified.

- DidReceiveAd: Called when SDK receives successful ad response from ad server.
- DidFailToReceiveAd: Called when request to ad server failed and passes the error to user.
- WillPresentScreen: Called when AdcashView is attached to the main window.
- WillLeaveApplication: Called when user clicks the ad and gets redirected to ad's URL.
- WillDismissScreen: Called when AdcashView is closed.

Objective-C SDK implementation can be improved in few ways:

First of all, if a developer wants to implement different ad types on the same UIViewController, they have to conform both banner and interstitial protocols as shown in Snippet 31.

```
@interface MainScreen: UIViewController <ADCBannerViewDelegate,
ADCInterstitialDelegate>
@end
```

Snippet 31: Conforming ADCBannerViewDelegate and ADCInterstitialDelegate.

This implementation leads to boilerplate code in the class, as seen in Snippet 32. Since it is not possible to have two ads presenting at the same time, we can drop the class instances as well, which will allow us to create unified protocol, that can be used in all ad formats.

```
-(void)bannerViewDidReceiveAd:(ADCBannerView *)bannerView {
   NSLog(@"Banner received");
}

-(void)interstititalDidReceiveAd:(ADCInterstitial *)interstitial {
   NSLog(@"Interstitial received");
}
```

Snippet 32: bannerViewDidReceiveAd: and intertitialDidReceiveAd methods.

With the new implementation presented in Snippet 33, developers only need to conform single protocol.

```
protocol AdcashProtocol {
   func adReceived()
   func adFailedToReceive(error: Error)
   func adWillPresentScreen()
   func adWillLeaveApp()
   func adWillDismissScreen()
   func adRewarded(name: String, amount: Int) // Introduced with AdcashRewardedVideo
}
```

Snippet 33: AdcashProtocol methods.

Making protocol functions optional was possible by adding the @optional keyword in Objective-C. In Swift same functionality can be achieved with adding protocol extensions as seen in Snippet 34 and provide default implementation for protocol functions.

```
extension AdcashProtocol {
   func adReceived() {}
   func adFailedToReceive(error: Error) {}
   func adWillPresentScreen() {}
   func adWillLeaveApp() {}
   func adWillDismissScreen() {}
}
```

Snippet 34: Default implementations for AdcashProtocol.

This approach makes AdcashProtocol optional-like by letting developers to override the functions, if they want to take advantage of it.

On the other hand, since the adRewarded function is mandatory to implement for rewarded videos, we extend the protocol as seen in Snippet 35.

```
extension AdcashProtocol where Self: AdcashRewardedVideo {
   func adRewarded(name: String, amount: Int) {
       fatalError("adRewarded:name:amount function should be implemented.")
   }
}
```

Snippet 35: AdcashProtocol extension for AdcashRewardedVideo.

So the default implementation will give a fatal error and force developers to override it.

4.3 Ad Type Specific

This section gives an overview about the architecture and implementation of the ad classes of Adcash Swift SDK.

4.3.1 AdcashBanner

Object-value separation introduced in AdcashView enabled us to have lighter ad classes. ADCBannerView class in Adcash Objective-C SDK was responsible of keeping track of every referenced item related to banner ad. These references include banner reloader, timer, visibility observer, API response and so on. With the current implementation, complexity is removed and AdcashBanner only responsible for making requests and populating the AdcashView's StateValue with the response.

To implement the AdcashBanner, developers must import the module as seen in Snippet 36.

```
import "AdcashSwift"
```

Snippet 36: Importing Adcash Swift SDK.

If they want to receive ad events, they can do it by conforming the Adcash Protocol as presented in Snippet 37.

```
class YourClass: UIViewController, AdcashProtocol {
   // ...
}
```

Snippet 37: Conforming AdcashProtocol.

As the last step, they have to implement the code below into their viewDidLoad() method of their UIViewController subclass as implemented in Snippet 38.

```
var banner: AdcashBanner!
banner = AdcashBanner(zoneID: "your-zone-id", viewController: self)
banner.translatesAutoresizingMaskIntoConstraints = false
self.view.addSubview(banner)
banner.delegate = self
self.view.addConstraints(NSLayoutConstraint.constraints(
   withVisualFormat: "V:[banner(\(AdcashBannerSize))]|",
   options: NSLayoutFormatOptions(rawValue: 0),
   metrics: nil,
   views: ["banner": banner!]))
self.view.addConstraints(NSLayoutConstraint.constraints(
   withVisualFormat: "H:|[banner]|",
   options: NSLayoutFormatOptions(rawValue: 0),
   metrics: nil,
   views: ["banner": banner!]))
banner.load()
```

Snippet 38: Initialization and loading of AdcashBanner.

If they conformed AdcashProtocol, they will also have access to delegate methods.

With the initialization of AdcashBanner as shown in Figure 9, SDK initializes AdcashView with the type of banner and sets its colour to transparent. Later on when load function called, SDK reads the parameters from AdcashConstants and uses them in the request to ad server. Received response is saved into StateValue and passed to AdcashView. When StateValue pushed to AdcashView, timer starts for refresh rate and extension for visibility starts listening for visibility change to pause or restart the timer.

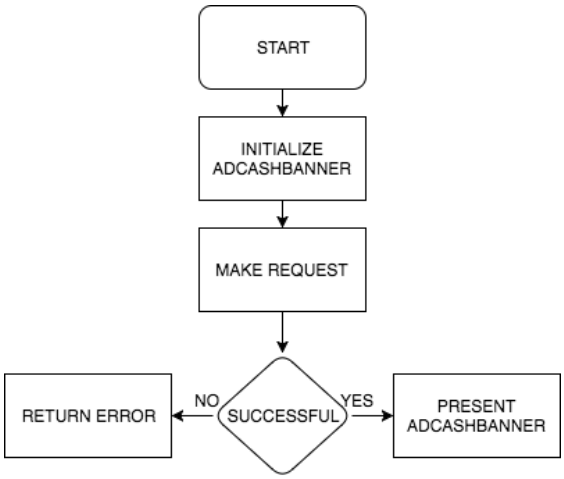

Figure 9: Initialization flow of AdcashBanner.

In Adcash Objective-C SDK, ADCBannerView was holding reference to instance of ADCBannerReloader class that is holding references to ADCBannerVisibilityObserver and ADCTimerManager. With Swift SDK, visibility moved into a AdcashBanner extension and class stayed as a separate class. The reason to keep timer as a class is, it's need to refer system objects for counting time. Since its visibility logic is moved to extension, ADCBannerReloader class is completely removed and all functionality of it handled by a single reload function.

New implementation of visibility logic is dependent on four values, as it was on previous implementation:

- appDidEnterBackground: Called when application switches to background state.
- appWillEnterForeground: Called when application switches to active state.
- isHidden: A default property value that determines whether the view is hidden.
- hasWindow: This property is nil if the view has not yet been added to main window.

AdcashBanner observes any change on these values to update the isVisible value that is presented in Snippet 39.

```
var isVisible: Bool? = true {
   didSet{
       If isVisible == false {
           bannerTimer.pause()
       } else {
           bannerTimer.continue()
       }
   }
}
```

Snippet 39: isVisible variable.

4.3.2 AdcashInterstitial

When publisher wants to use an interstitial in their application, they have to do it with implementing the code snippets shown below.

As a first step, they have to import the module and conform the protocol (optional) as shown in Snippet 40 and Snippet 41 respectively.

```
import AdcashSwift
```

Snippet 40: Importing Adcash Swift SDK.

```
class YourClass: UIViewController, AdcashProtocol {
   // ...
}
```

Snippet 41: Conforming AdcashProtocol.

After that publisher can initialize and load the interstitial as shown in Snippet 42 and after successful loading, they can present it to screen as show in Snippet 43.

```
var interstitial: AdcashInterstitial!
interstitial = AdcashInterstitial(zoneID: "your-zone-id")
interstitial.delegate = self // Available if AdcashProtocol is conformed
interstitial.load()
```

Snippet 42: Initialization and loading of AdcashInterstitial.

```
interstitial.present(fromVC:self)
```

Snippet 43: Presenting AdcashInterstitial to the screen.

With the initialization of AdcashInterstitial as shown in Figure 10, SDK initializes a full screen AdcashView. After load function is called, SDK reads the parameters from AdcashConstants and uses them in request to ad server. If request is ended with successful response, SDK saves response in StateValue and passes it to AdcashView. When developer calls the present function, AdcashView attaches to screen and presents the ad.

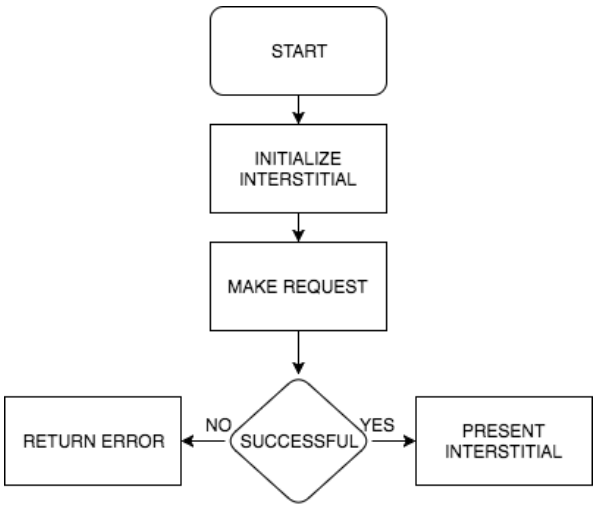

Figure 10: Initialization flow of AdcashInterstitial.

4.3.3 AdcashRewardedVideo

Rewarded videos are full screen video ads where users watch the video ad in exchange of in-game rewards.

When developers want to implement AdcashRewardedVideo, they have to follow the steps presented in snippets below:

First of all, they have to import the SDK as shown in Snippet 44.

```
import AdcashSwift
```

Snippet 44: Importing Adcash Swift SDK.

Secondly, they have to conform AdcashProtocol as seen in Snippet 45.

```
class YourClass: UIViewController, AdcashProtocol {
   //...
}
```

Snippet 45: Comforming AdcashProtocol.

As the third step, they have to declare an AdcashRewardedVideo variable in their class an initialize it as show in Snippet 46.

```
var rewarded: AdcashRewardedVideo!
rewarded = AdcashRewardedVideo(zoneID:"your-zone-id")
rewarded.delegate = self
```

Snippet 46: Initializing AdcashRewardedVideo.

After AdcashRewardedVideo loaded, they can present it to user with the function presented in Snippet 47.

```
rewarded.playFrom(viewController:self)
```

Snippet 47: Presenting AdcashRewardedVideo.

Since conforming AdcashProtocol is mandatory for AdcashRewardedVideo, developers also have access to delegate methods, presented in Snippet 48.

```
func adReceived()
func adFailedToReceive(error: Error)
func adWillPresentScreen()
func adWillDismissScreen()
func adWillLeaveApplication()
func adRewarded(name: String, amount: Int) // required
```

Snippet 48: Available AdcashProtocol methods for AdcashRewardedVideo.

As seen in Snippet 46, AdcashRewardedVideo doesn't have an explicit method for loading. The reason behind this is to have the video cached in the memory with initial initialization. So when the publisher wants to present the video, it will not take time of video player to buffer the video file. Because of this motivation, AdcashRewardedVideo designed to have three components as AdcashParser, AdcashCache and Adcash Video Player.

4.3.3.1 AdcashParser

After the initialization of AdcashRewardedVideo, class makes a request to Adcash Ad Server and receives VAST response as seen in Appendix I. This is where AdcashParser

takes the response and parses it in VideoStateValue (Snippet 49) struct for AdcashRewardedVideo class.

```
struct VideoStateValue {
    ...
    var VASTAdTagURI = (set: Bool, value: String)
    var videoLow = (set: Bool, value: String)
    var videoMid = (set: Bool, value: String)
    var videoHigh = (set: Bool, value: String)
    var clickThrough = (set: Bool, value: String)
    var duration = (set: Bool, value: String)
    ...
}
```

Snippet 49: VideoStateValue struct.

AdcashParser is a class that conforms Apple's built-in XMLParserDelegate class protocol from Foundation framework to parse the XML files. Optional functions that are implemented from protocol are listed below with explanations:

Table 4: Implemented XMLParserDelegate methods in AdcashParser.

1	`func parserDidStartDocument(XMLParser)`
2	`func parser(XMLParser, didStartElement elementName: String, namespaceURI: String?, qualifiedName qName: String?, Attributes attributeDict: [String: String] = [ : ])`
3	`func parser(XMLParser, foundCDATA: Data)`
4	`func parser(XMLParser, parseErrorOccurred: Error)`
5	`func parser(XMLParser, foundCharacters: String)`
6	`func parserDidEndDocument(XMLParser)`

Instance method presented in the first row of Table 4 is called by parser object when it begins parsing a document. Within this method, we initialize the VideoStateValue to save the values parsed from XML file. Method in second row is sent by parser object to its delegate when it encounters a start tag for a given element. Within this method we compare ElementName with tuples' (tuples are not supported in Objective-C) strings from VideoStateValue. When elementName and tuple's string match, we set that tuple's Boolean value to true. Instance method in third row is called by parser object when it encounters a CDATA[21] block. As it stated in Apple's documentation[22], this method passes the contents of blocks in NSData[23] object and the parser ignores the CDATA block contents. In VAST file, all HTTP links are presented in CDATA blocks so characters such as "/" can escape the parser. To get the links, we convert the CDATA blocks to string with UTF-8[24] encoding. Later on we save the converted string to its tuple and set the Boolean to false. Instance method presented in fourth row is sent when parser object encounters a fatal error. At this point we are not interested with the reason of error since it's from an external file. So we simply call the AdcashProtocol's method for adFailedToReceive. If parser object encounters a string representing all or part of the characters of the current element, method in the fifth row is called. This is the method that is used to catch the values such as video

[21] https://en.wikipedia.org/wiki/CDATA
[22] https://developer.apple.com/documentation/foundation/xmlparserdelegate/1407687-parser
[23] https://developer.apple.com/documentation/foundation/nsdata
[24] https://en.wikipedia.org/wiki/UTF-8

duration, that are not links. Instance method presented in the last row of Table 4 is sent by parser object when it has completed parsing successfully.

Adcash Ad Server has API integrations with third party video ad providers. When we receive the video ads from them, Adcash Ad Server puts them into wrapper and sends it to SDK. To access the wrapped XML, we parse the wrapper XML and look for the tag "VASTAdTagURI" which is URL of ad tag of third party ad server. Within parserDidEndDocument method presented in Table 4, we look if the "VASTAdTagURI" exists. If it does, it parses the XML in that URL. After parsing is completed and VideoStateValue is ready to be used by video player.

4.3.3.2 AdcashCache

Adcash Swift SDK introduced another feature for video ads. To improve the SDK performance by reducing latency, videos are being cached in disk. With the caching logic flow presented in Figure 11.

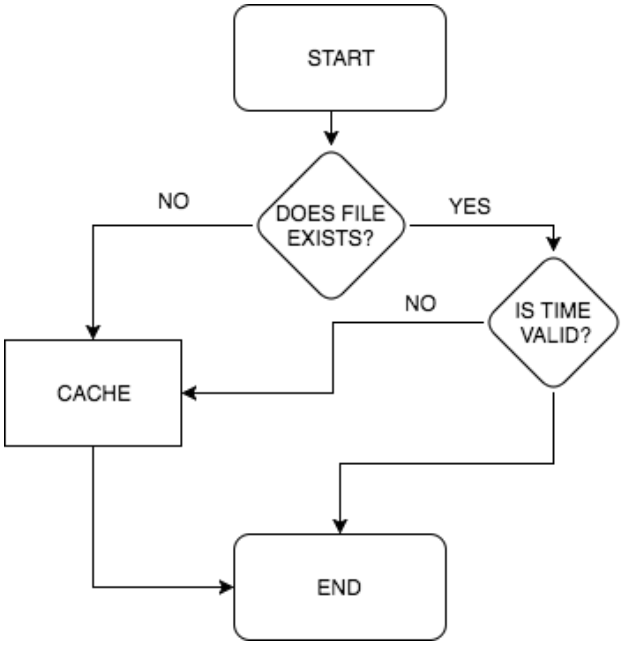

Figure 11: AdcashCache workflow.

When AdcashRewardedVideo makes a request to Adcash Ad Server with zone ID, we check if there is a cached response and video file for that zone. If there is a cached file saved with the value of zone ID and its time is valid (1 hour in our case), we let the video player use the cached video and response.

If there is no cached video or time is invalid, SDK makes a new request, parses the response and decide the video quality based on previous download time and quality.

The motivation behind the quality selection is to provide best option as quick as possible, when devices have poor internet connection. Quality selection is made based on the values that are shown in Table 5.

Table 5: Video quality selection.

Download Time x (s)	Downloaded Quality	Selected Quality
x ≤ 2s	Low	High
2s < x ≤ 3s	Low	Mid
x > 3s	Low	Low
x ≤ 3s	Mid	High
3s < x ≤ 5s	Mid	Mid
x > 5s	Mid	Low
x ≥ 10s	High	Low
5s ≤ x < 10s	High	Mid
x < 5s	High	High
NA	NA	Low

After deciding the quality, link corresponding to video quality gets downloaded and saved as "zone-id".mp4 to application's file directory.

And last, AdcashCache saves the struct show in Snippet 50 to UserDefaults[25] for future access.

```
struct AdcashCache {
   var stateValue: VideoStateValue
   var info = (time: Double, quality: String)
}
```

Snippet 50: AdcashCache struct.

4.3.3.3 Adcash Video Player

Adcash Video Player is designed to cover two requirements:

- Capable of working with VAST files
- Restricting controls given to the users.

Adcash Video Player is created using AVFoundation[26] and AVKit[27] frameworks by Apple. AVFoundation is used for working with video file and AVKit is used for user interface controls.

After video is initialized, video player reads the VideoStateValue from UserDefaults and sets the video file as the player item. Instead of feeding player with the mp4 file or with the source URL directly, by adding them as a player item gives us the capability to observe player item events such as its status.

With overriding observeValue function as seen in Snippet 51 and filtering the changes for AVPlayerItem status, we are handling errors caused by the media item. When some error happens, we make the close button visible so users can close it instead of killing the application to get rid of the stuck video player.

[25] https://developer.apple.com/documentation/foundation/userdefaults
[26] https://developer.apple.com/documentation/avfoundation
[27] https://developer.apple.com/documentation/avkit

```
override public func observeValue(forKeyPath keyPath: String?, of object: Any?,
change: [NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?) {
        guard context == &playerItemContext else {
            return
        }
        if keyPath == #keyPath(AVPlayerItem.status) {
            let status: AVPlayerItemStatus

            if let statusNumber = change?[.newKey] as? NSNumber {
                status = AVPlayerItemStatus(rawValue: statusNumber.intValue)!
            } else {
                status = .unknown
            }

            switch status {
            case .readyToPlay:
                //ready for play, do nothing
                break
            case .failed, .unknown:
                self.skipButton.isHidden = false
                break
            }
        }
    }
```

Snippet 51: Overwritten observeValue function.

There are two custom and four built-in functions that video player is observing:

- applicationWillResignActive[28]
- applicationWillEnterForeground[29]
- applicationDidBecomeActive[30]
- orientationDidChange
- watchTheTime
- AVPlayerItemDidPlayToEndTimeNotification[31]

applicationWillResignActive is called when the application is about to go inactive state. It can be upcoming interaction such as call or simply the closing the application. If this transition happens while video is playing, we pause the player.

applicationWillEnterForeground is called when the application about to go an active state. When the user returns back to the app, we continue playing the video, from where it's left off.

applicationDidBecomeActive is called when the application changed its state to active state. In this method we continue playing the video, if there is any paused. The difference between *applicationWillEnterForeground* and *applicationDidBecomeActive* is, former is triggered with opening Control Center[32] in iOS while latter is not. This causing the player to keep playing the video on background, while Control Center is open.

[28] https://developer.apple.com/documentation/uikit/uiapplicationdelegate/1622950-applicationwillresignactive
[29] https://developer.apple.com/documentation/uikit/uiapplicationdelegate/1623076-applicationwillenterforeground
[30] https://developer.apple.com/documentation/uikit/uiapplicationdelegate/1622956-applicationdidbecomeactive
[31] https://developer.apple.com/documentation/avfoundation/avplayeritemdidplaytoendtimenotification
[32] https://en.wikipedia.org/wiki/Control_Center_(iOS)

orientationDidChange is set to observe UIDeviceOrientationDidChangeNotification[33]. When the notification received, we check the orientation information and update user interface controls for the new orientation.

watchTheTime is not an observable method itself but it is used within addPeriodicTimeObserver[34] method directly attached to player, to be informed about video's progress. As it stated in its documentation, this method requests the periodic invocation of a given block during playback to report changing time.

We used the block time for periodic observer to be 1 second. Therefore we can update the timer label and calculate the quarters to make request to necessary links as shown in Snippet 52.

```
func watchTheTime(current: TimeInterval, total: Int) {
    countdown.text = String(total - Int(current)) // Updating countdown label
    if current >= Double(total)*0.25 && stateValue.first.fired == false {
        fire(toLink:stateValue.first.link)
        stateValue.first.fired = true
    }
    if current >= Double(total)*0.5 && stateValue.mid.fired == false {
        fire(toLink:stateValue.mid.link)
        stateValue.mid.fired = true
    }
    ...
}
```

Snippet 52: watchTheTime function.

AVPlayerItemDidPlayToEndTimeNotification is the last notification we are observing for the video player. Notification is posted when the item has played to its end time. Here we have to possible paths, as presented in Figure 12.

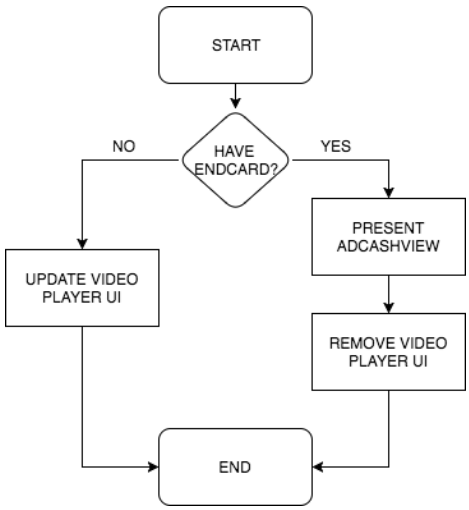

Figure 12: End card flow.

When a user has completed watching a video, end cards allow the user to interact with the ad and easily connect with the advertiser. In simpler words, they are AdcashInterstitial's shown right after video is ended.

[33] https://developer.apple.com/documentation/uikit/uideviceorientationdidchangenotification
[34] https://developer.apple.com/documentation/avfoundation/avplayer/1385829-addperiodictimeobserver

End cards are presented in two different ways within VAST files. Either as a static image link or a HTTP file. To check if end card exists, we look inside the StaticResource or HTMLResource in VideoStateValue. If one of them are not nil, we use that value for feeding the AdcashInterstitial as a StateValue. As SDK presents the end card as seen in Figure 13, it removes the video player from screen.

Figure 13: End card example.

If there is no end card presented in VAST file, video rewinds back to first frame and all video player controls becomes visible with addition of replay and download buttons as seen in Figure 14.

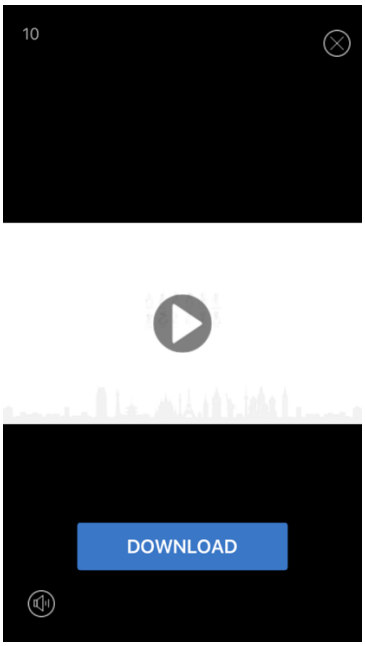

Figure 14: Replay screen.

Adcash Video Player hides default playback controls by setting showsPlaybackControls[35] property to false. Instead we use a transparent UIView instance over on top of Adcash Video Player with our visual elements attached on it. There are five visual elements as seem in Table 6:

Table 6: Visual elements of Adcash Video Player.

Name	Description
Countdown	Updates its text every second with the remaining seconds of video.
Mute / Unmute Button	Mutes or unmutes the video.
Action Button	Redirects users to advertisement's click through link.
Replay Button	Restarts video when clicked.
Close Button	Removes the ad from screen.

4.3.4 AdcashNative

Native advertisement is a form of paid media where the ad experience follows the natural form and function of the user experience in the platform where it's placed. As seen in Figure 15, good native ads should match the visual design of the experience they live within and behave consistently with the functionality just like natural content.

Figure 15: Facebook native ads in *Cut the Rope* and *Shazam*.

Native ads are recently introduced in Adcash and still is in beta phase. Therefore, only some publishers are able to try and test it. With the current solution, SDK makes a request to Adcash Ad Server, receives native ad in JSON format forwards it to publishers, so they can design the ad experience by themselves.

[35] https://developer.apple.com/documentation/avkit/avplayerviewcontroller/1615824-showsplaybackcontrols

When the publisher wants to use native ads in their applications, they must follow the steps presented in snippets below:

First of all, to access the AdcashNative class, they have to import the module as seen in Snippet 53.

```
import "AdcashSwift"
```

Snippet 53: Importing Adcash Swift SDK.

As a second step, if they want to receive ad events, they have to conform AdcashProtocol as presented in Snippet 54.

```
class YourClass: UIViewController, AdcashProtocol {
   // ...
}
```

Snippet 54: Conforming AdcashProtocol.

To receive the ad, they should initialize the AdcashBanner instance as seen in Snippet 55.

```
var native: AdcashNative!
native = AdcashNative(zoneID: "your-zone-id")
native.delegate = self  //optional
```

Snippet 55: Initialization of AdcashNative.

After the ad is successfully loaded, AdcashNative instance will be filled with the ad information as seen in Table 7.

Table 7: AdcashNative properties and their accessors.

Name	Function	Explanation
Title	getAdTitle()	Title of the ad.
Ad Description	getAdDescription()	Description of the ad.
Rating	getAdRating()	Rating of the product from app store reviews.
Icon	getAdIconURL()	Icon URL of the product.
Image	getAdImageURL()	Main image URL of the product.
Button Text	getAdButtonText()	Action button text such as “Download Now”.

Publishers should also implement the redirect and impressions by themselves. It means that when the ad is visible on the screen, they should call the trackImpression() which will trigger the link to record the impression for our ad server. For redirecting links, they should modify their content to use openClick() function whenever users touches the ad, which will redirect them to products page for action.

Since it totally depends on the design of the application, there are no common guidelines about how to implement it. Most important part other than design is monetization. Publishers should be careful on tracking impressions or redirecting clicks so they will not lose any money with such issues.

5 Conclusion

This chapter summarizes the work done with Adcash Objective-C and introduces future plans.

5.1 Summary

The thesis started with detailed analysis of Adcash Objective-C SDK and its components. After the analysis of each component, disadvantages of implementations are investigated.

An existing third-party solution for MRAID was rewritten for Swift 3.

For the networking component of the Adcash Swift SDK, an analysis made between popular options, pros and cons discussed and decision made.

Two major and popular ad formats that were not supported in Adcash Objective-C SDK are now supported in Adcash Swift SDK.

End-to-end video player solution for VAST files built.

With taking advantage of Swift's support of value types, we separated the application into two layers: the object layer and the value layer. The object layer consists objects that are needed for system events and computation of the value while the value layer consists the data from the received advertisements. This led us to push all ad logic to value layer which helps us to address root of the issues very swiftly. On the other hand, this separation opened up possibilities to move SDK to more functional programming approaches.

5.2 Future Plans

Initial idea of developing Adcash Swift SDK was to trim third-party libraries as much as possible. This has been achieved for MRAID library but not for networking component. The future plan for this component is to move it in a wrapper class, so we can change the library or develop our own without disrupting the SDK flow. Another idea for networking component is to, develop it by using Protocol Oriented Programming approach [18]. With this approach all networking functionality can be isolated from class infrastructure and all calls can be made with a single line expression.

Initial version of native ads has no restrictions and gives all information to developers. In coming versions, we are planning to isolate the click and impression URLs from the users and automate them, so monetization will not be impacted from wrong implementation or fraud.

The development of the Swift SDK has just been completed and only a minimal amount of testing has been conducted. Future work will go into conducting functional, performance and reliability testing of the SDK. This will happen as Adcash's customers start adopting the new SDK.

6 References

[1] H. Weber, „Apple announces 'Swift', a new programming language for OS X & iOS,“ 2 June 2014. [Online]. Available: https://venturebeat.com/2014/06/02/apple-introduces-a-new-programming-language-swift-objective-c-without-the-c/. [Accessed 8 August 2017].

[2] Alamofire Software Foundation, „AFNetworking,“ [Online]. Available: https://github.com/AFNetworking/AFNetworking. [Accessed July 2017].

[3] B. Copsey, “ASIHTTPRequest,” All Seeing, [Online]. Available: https://github.com/pokeb/asi-http-request. [Accessed July 2017].

[4] R. Hanson, „CocoaAsyncSocket,“ [Online]. Available: https://github.com/robbiehanson/CocoaAsyncSocket. [Accessed July 2017].

[5] Foursquare, „FSNetworking,“ [Online]. Available: https://github.com/foursquare/FSNetworking.

[6] The RestKit Project, „RestKit,“ [Online]. Available: https://github.com/RestKit/RestKit.

[7] Alamofire Software Foundation, [Online]. Available: https://github.com/AFNetworking/AFNetworking/wiki/Introduction-to-AFNetworking. [Accessed July 2017].

[8] Nexage Inc., „Nexage Integration SourceKit for MRAID,“ [Online]. Available: https://github.com/nexage/sourcekit-mraid-ios.

[9] Millennial Media, „Millennial Media to Acquire Nexage, a Leading Mobile SSP and Advertising Exchange,“ Millennial Media, 23 September 2014. [Online]. Available: http://www.millennialmedia.com/press/millennial-media-to-acquire-nexage-a-leading-mobile-ssp-and-advertising-exchange. [Accessed 23 July 2017].

[10] Interactive Advertising Bureau, „Digital Video Ad Serving Template (VAST) 2.0,“ Interactive Advertising Bureau, 1 March 2010. [Online]. Available: https://www.iab.com/guidelines/digital-video-ad-serving-template-vast-2-0/. [Accessed 9 August 2017].

[11] E. Gamma, R. Helm, R. Johnson ja J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software., Addison Wesley Professional, 1994.

[12] Alamofire Software Foundation, „Alamofire,“ July 2017. [Online]. Available: https://github.com/Alamofire/Alamofire.

[13] D. Duan, „Just,“ [Online]. Available: https://github.com/JustHTTP/Just. [Accessed July 2017].

[14] E. Nuñez, „Networking,“ [Online]. Available: https://github.com/3lvis/Networking. [Accessed Juy 2017].

[15] J. Lui, „Pitaya,“ [Online]. Available: https://github.com/johnlui/Pitaya. [Accessed July 2017].

[16] Bust Out Solutions Inc., „Siesta,“ [Online]. Available: https://github.com/bustoutsolutions/siesta/issues. [Accessed July 2017].

[17] A. Matuschak, „Controlling Complexity in Swift: Making Value Types Friends,“ 11 February 2015. [Online]. Available: https://academy.realm.io/posts/andy-matuschak-controlling-complexity/. [Accessed 11 July 2017].

[18] Apple, „Protocol-Oriented Programming in Swift,“ Apple, 2015. [Online]. Available: https://developer.apple.com/videos/play/wwdc2015/408/. [Accessed 2017].

Appendix

I. VAST Template

```
<?xml version="1.0" encoding="UTF-8"?>
<VAST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="3.0"
xsi:noNamespaceSchemaLocation="vast3.xsd">
 <Ad id="65217051">
   <InLine>
     <AdSystem version="1.0">Adcash</AdSystem>
     <Error>
     <![CDATA[http://adserver.com/&verrorcode=[ERRORCODE]]]>
     </Error>
     <Impression></Impression>
     <Extensions>
       <Extension name="RewardedVideo">
           <RewardedVideoOn><![CDATA[1]]></RewardedVideoOn>
           <VirtualCurrency><![CDATA[Lives]]></VirtualCurrency>
           <RewardPerView><![CDATA[12.000000]]></RewardPerView>
           <VirtualCurrencyImage><![CDATA[http://adserver.com/lives123.jpg]]></VirtualCurrencyImage
>
           <PreRollMessageOn><![CDATA[1]]></PreRollMessageOn>
           <PostRollMessageOn><![CDATA[1]]></PostRollMessageOn>
       </Extension>
     </Extensions>
     <Creatives>
       <Creative sequence="1">
         <Linear> // optional "Skip allowed" -> <Linear skipoffset="00:00:05">
           <Duration>00:00:10</Duration>
           <TrackingEvents>
             <Tracking
event="start"><![CDATA[http://adserver.com/&vau=[ASSETURI]&vprog=[CONTENTPLAYHEAD]]]></Tracking>
             <Tracking
event="skip"><![CDATA[http://adserver.com/&vau=[ASSETURI]&ve=skip&vprog=[CONTENTPLAYHEAD]]]></Track
ing>
             <Tracking
event="pause"><![CDATA[http://adserver.com/&vau=[ASSETURI]&ve=pause&vprog=[CONTENTPLAYHEAD]]]></Tra
cking>
             <Tracking
event="resume"><![CDATA[http://adserver.com/&vau=[ASSETURI]&ve=resume&vprog=[CONTENTPLAYHEAD]]]></T
racking>
           </TrackingEvents>
           <VideoClicks>
             <ClickThrough><![CDATA[http://adserver.com/?event=videoclick]]></ClickThrough>
           </VideoClicks>
           <MediaFiles>
             <MediaFile delivery="progressive" bitrate="596" width="426" height="240"
type="video/mp4"><![CDATA[http://cdn.com/ban/lo.mp4]]></MediaFile>
             <MediaFile delivery="progressive" bitrate="1128" width="854" height="480"
type="video/mp4"><![CDATA[http://cdn.com/ban/me.mp4]]></MediaFile>
             <MediaFile delivery="progressive" bitrate="1628" width="1280" height="720"
type="video/mp4"><![CDATA[http://cdn.com/ban/hi.mp4]]></MediaFile>
           </MediaFiles>
         </Linear>
         <CompanionAds>
           <Companion id="14184407" width="768" height="1024">
             <HTMLResource><![CDATA[http://adserver.com/htmlresourcefile.htm]]></HTMLResource>
             <TrackingEvents>
               <Tracking
event="creativeView"><![CDATA[http://adserver.com/?event=endcardview]]></Tracking>
             </TrackingEvents>
             <CompanionClickThrough><![CDATA[http://www.adcash.com]]></CompanionClickThrough>
           </Companion>
         </CompanionAds>
       </Creative>
     </Creatives>
   </InLine>
 </Ad>
</VAST>
```

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Mert Celik**,

herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Reimplementing the Adcash Software Development Kit in Swift,

supervised by Marlon Dumas, PhD,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **14.08.2017**