iOS Code Integration

In order to use header bidding on Mobile, you first need to have an Admax Prebid server account. Once your account is set up, you can include the iOS Admax header bidding mobile SDK in your app by either using pods or using framework bynary file that we can provide.

If you are using pods, you can include our SDK to your app by adding our framework dependency in your podfile:

pod 'AdmaxPrebidMobile' , '1.1.21'

Also ensure that you set iOS platform to 9.0, setting the platform to an earlier version might return unexpected results:

platform :ios, '9.0'

Finally configure Cocoapods to go back using the Git spec repo by adding to the top of your Podfile the following:
source 'https://github.com/CocoaPods/Specs.git'

Google Ad Server

If this header bidding integration is done in conjunction with a google Ad Server, you also need to add Google Mobile Ads SDK to your application. You can check Google's developer website and follow the instructions to integrate their Mobile Ads SDK (https://developers.google.com/admob/ios/quick-start). We also support MoPub AdServer and Smart AdServer

Smart Ad Server.

If this header bidding integration is done in conjunction with a Smart Ad Server, you also need to add Smart Display SDK to your application. You can check Smart's documentation and follow the instructions to integrate their Display SDK (http://documentation.smartadserver.com/displaySDK/ios/gettingstarted.html).

If using Google backfill in conjunction with Smart Ad Server, there is no need to add GoogleMobileAdsSdk dependency in the Podfile, as it is already included in our SDK.

In addition, in order to enable header bidding, Smart Ad Server SDK requires the app that wants to add third party demand to implement SASBidderAdapterProtocol in order to create an iOS custom in-app bidding adapter (http://documentation.smartadserver.com/displaySDK/ios/inappbidding/customadapter.html)

Here is the code for the AdmaxBidderAdapter to add in your app in order to enable in-app bidding with Smart Ad Server :

```
import Foundation

import SASDisplayKit

import AdmaxPrebidMobile
```

```
class SASAdmaxBidderAdapter: NSObject, SASBidderAdapterProtocol,
UpdatableProtocol {

    var creativeRenderingType: SASBidderAdapterCreativeRenderingType =
SASBidderAdapterCreativeRenderingType.typePrimarySDK

    var adapterName: String = "SASAdmaxBidderAdapter"

    var winningSSPName: String = ""

    var winningCreativeID: String = ""

    var hbCacheID: String = ""

    var price: Float = 0

    var currency: String = "USD"

    var dealID: String? = ""

    var targetingMap: String = ""

    var admaxAdUnit: AdUnit


    init(adUnit: AdUnit) {

        admaxAdUnit = adUnit

    }


    public func update(keywords: [String: String]) {

        winningSSPName = keywords["hb_bidder"]!

        winningCreativeID = keywords["hb_cache_id"]!

        hbCacheID = keywords["hb_cache_id"]!

        price = Float(keywords["hb_pb"]!)!

        targetingMap = stringify(keywords: keywords)

    }


    func primarySDKDisplayedBidderAd() {

        print("primarySDKDisplayedBidderAd called")

        admaxAdUnit.sendBidWon(bidWonCacheId: hbCacheID)

    }


    func primarySDKClickedBidderAd() {

        print("primarySDKClickedBidderAd called")

    }


    func primarySDKLostBidCompetition() {

        print("primarySDKLostBidCompetition called")
```

```
    }

    func bidderWinningAdMarkup() -> String {
        let appId: String = Bundle.main.bundleIdentifier!
        let adm: String = "<script src =
\"https://cdn.admaxmedia.io/creative.js\"></script>\n" +
            "<script>\n" +
            "  var ucTagData = {};\n" +
            "  ucTagData.adServerDomain = \"\";\n" +
            "  ucTagData.pubUrl = \"" + appId + "\";\n" +
            "  ucTagData.targetingMap = " + targetingMap + ";\n" +
            "\n" +
            "  try {\n" +
            "    ucTag.renderAd(document, ucTagData);\n" +
            "  } catch (e) {\n" +
            "    console.log(e);\n" +
            "  }\n" +
        "</script>"
        return adm
    }

    private func stringify(keywords: [String: String]) -> String {
        let n: Int = keywords.count - 1
        var i: Int = 0
        var keywordsString = "{"
        for (key, value) in keywords {
            keywordsString.append("'")
            keywordsString.append(key)
            keywordsString.append("':['")
            keywordsString.append(value)
            keywordsString.append("']")
            if (i < n) {
                keywordsString.append(",")
            }
            i += 1
        }
        keywordsString.append("}")
```

```
        return keywordsString
    }
}
```

Supported Ad Units

Admax supports both **banner** and **interstitial** ad units. The SDK exposes the following classes: **BannerAdUnit** and **InterstitialAdUnit**. See the detailed API below for details on each Ad Unit type, the parameters and methods associated with each one of them.

Admax iOS Header Bidding API

Admax iOS Header Bidding API helps publishers implementing header bidding in their mobile applications. The elements in the API will allow them to participate in a header bidding auction and communicate with your ad server to display a creative. The API supports banner and interstitial creatives.

The key features of Admax iOS API are:

- As a publisher, keywords can be attached to each adUnit to improve targeting.
- In addition, ad units implement and support their own auto refresh, so there is no longer need to support adServer refresh.
- Admax iOS API provides a powerful Analytics Adapter that feeds our dashboards in real time with all the data of the header bidding auctions and winners
- Admax iOS API finally provides clear result codes that detail the response of the Header Bidding demand fetch request

Admax iOS API supports instantiation of the following objects:

- Global Prebid Mobile Ad Unit Settings
- Banner Ad Unit
- Interstitial Ad Unit

PrebidMobile Object

The PrebidMobile object is used to apply global settings to the app.

- Object
  - PrebidMobile
- Properties
  - loggingEnabled
  - admaxExceptionLogger
  - prebidServerAccountId
  - shareGeoLocation
  - prebidServerHost
  - prebidAnalyticsServerHost

Properties

loggingEnabled

For debugging purposes, Admax SDK’s logs can be enabled. By default, logging is set to false.

admaxExceptionLogger

Prebid Mobile allows the application using the SDK to receive the exceptions logged by our SDK. In order to retrieve those exceptions and forward them to a monitoring system as Crashlytics, the application needs to implement the AdmaxExceptionLogger protocol and initialize the SDK with the object implementing the exception logger interface.

prebidServerAccountId

Define the Admax Prebid server account ID. We provide each publisher with a distinct account ID.

prebidServerHost

Define Admax Prebid Server host with which the SDK will communicate. Default = PrebidHost.Admax

prebidAnalyticsServerHost

Define Admax Prebid Server analytics host with which the SDK will communicate. Default = PrebidAnalyticsHost.AdmaxAnalytics

shareGeoLocation

If this flag is true and the app collects the user's geographical location data, Admax iOS SDK will send the user's geographical location data to Prebid Sever. If this flag is False or the app does not collect the user's geographical location data, Prebid Mobile will not populate any user geographical location information in the call to Prebid Server. Default = false

Example Swift

```
Prebid.shared.prebidServerAccountId = "0dfe3a52-aeb2-4562-bdea-31bd2d69f214"
Prebid.shared.shareGeoLocation = true
Prebid.shared.prebidServerHost = PrebidHost.Admax
Prebid.shared.prebidAnalyticsServerHost = PrebidAnalyticsHost.AdmaxAnalytics
```

Example Objective C

```
Prebid.shared.prebidServerAccountId = @"0dfe3a52-aeb2-4562-bdea-31bd2d69f214";
Prebid.shared.prebidServerHost = PrebidHostAdmax;
Prebid.shared.shareGeoLocation = true;
```

AdUnit Object

The AdUnit object is an abstract object that cannot be instantiated. Use the BannerAdUnit or InterstitialAdUnit object to create and configure the desired type of ad unit within your app.

- Object
  - AdUnit
- Methods
  - fetchDemand
  - addUserKeyword
  - addUserKeywords
  - removeUserKeyword
  - clearUserKeywords
  - setAutoRefreshMillis
  - startAutoRefresh
  - stopAutoRefresh
  - sendBidWon

- setGamAdUnitId
- getGamAdUnitId

Object

AdUnit

AdUnit properties and methods are inherited by BannerAdUnit and InterstitialAdUnit.

Parameters

- *configId*: String containing the Admax Prebid Server configuration ID
- *adType*: BANNER or INTERSTITIAL. This value will be set by the object based on which type of ad unit object you create

Properties

- *configId*: Admax Prebid Server configuration ID
- *adType*: BANNER or INTERSTITIAL
- *periodMillis*: Integer defining the refresh time in milliseconds. Default = 0, meaning no auto refresh
- *keywords*: ArrayList containing keys and values
- rootViewController: UIViewController, required to be set when showing google ads when using backfill

Methods

fetchDemand

Trigger a call to Admax Prebid Server to retrieve demand for this Prebid Mobile ad unit.

Parameters

- *adObj*: bid request object
- *onCompleteListener*: listener object

addUserKeyword

Add a single key-value pair.

Parameters

- *key*: String containing the key
- *value*: String containing the value

addUserKeywords

Define multiple values for a single key.

Parameters

- *key*: String containing the key

- *values*: String array containing the list of values for the key

removeUserKeyword

Remove a key and all its associated values from a given Prebid Mobile ad unit.

Parameters

- *key*: String containing the key you want to remove

clearUserKeywords

Clear all key-value combinations from the Prebid Mobile ad unit.

Parameters

None

setAutoRefreshMillis

If set on a given Prebid Mobile ad unit, the *fetchDemand* function will be called every *periodMillis* until *stopAutoRefresh* is called. Each call to *fetchDemand* will invoke the *onComplete* function. This refresh only belongs to Admax iOS SDK and not to any ad server refresh process. Thus, the adserver refresh should be turned off.

Parameters

- *periodMillis*: Integer defining the refresh time in milliseconds

startAutoRefresh

Starts the auto-refresh behavior for a given Prebid Mobile ad unit.

Parameters

None

stopAutoRefresh

Stops the auto-refresh behavior for a given Prebid Mobile ad unit. If no auto-refresh behavior has been set, *stopAutoRefresh* will be ignored.

Parameters

None

sendBidWon

Trigger a call to Admax analytics Adapter to send the bid that has won the auction using its cacheId header bidding key. A bid is considered as won when our Google Ad Server creative is served, it will trigger an Admob event with the header bidding cacheId of the winning bid impression. Therefore, this method needs to be called as a callback to an app event within a Google Ad Manager

ad view. Hence, we use Google Mobile Ads SDK setAppEventListener method to add en event listener where our sendBidWon method will be called to send analytics data to our backend.

Parameters

- *hbCacheId*: winning header bidding bid cacheId key
- *eventName*: if this optional parameter is set, a check is made on the *eventName* to make sure it is a *bidWon* event before sending the *bidWon* Analytics event

setGamAdUnitId

Set GAM adUnit Id. Set only when using Google as backfill.

Parameters

gamAdUnitId: String, GAM adUnit Id

getGamAdUnitId

Get GAM AdUnit Id.

Parameters

None

BannerAdUnit Object

Use the BannerAdUnit object to create and configure a banner ad unit in your app.

Object

BannerAdUnit

Create a new Banner Ad Unit associated with a banner size and Admax Prebid Server configuration ID that is provided via our interface https://pbsadmin.admaxmedia.io.

Parameters

- *configId*: String, Admax Prebid Server configuration ID
- *width*: Integer, width of the ad unit
- *height*: Integer, height of the ad unit

Methods

BannerAdUnit inherits all methods from the AdUnit Object. It also includes the following additional methods:

addAdditionalSize

Add an additional banner size to the Admax Prebid Mobile ad unit. Banner ad units must be associated with one or more sizes. If Google backfill is enabled with Smart ad server, this method automatically converts the size to Google AdSize and ads it to Google requests. So no need to call

addAdditionalGamSize. In case the adserver is Google, this method should be used only (no need to use *addAdditionalGamSize*.

Parameters

width: integer
height: integer

addAdditionalGamSize

Add an additional GAM banner size to the Admax Prebid Mobile ad unit. This method is only used when using Google as backfill and when required to add a particular Google size to the request which is not present in *addAdditionalSize*. Sizes declared in this method, should be different from those in *addAdditionalSize* which have already been added to the Google request.

Parameters

size: GADAdSize, Google specific ad size

setGamBannerAdListener

Sets GAM ad listener. Used when using Google as backfill in order to set callbacks *onAdLoaded* and *onAdfailedToLoad* in order to retrieve for instance final ad size when using multisize requests.

Parameters

adListener: GamBannerAdListener, Admax protocol which allows the client to implement the following methods: *onAdLoaded*(size: GADAdSize) and *onAdFailedToLoad*(error: GADRequestError).

getGamBannerAdListener

Gets GAM ad listener.

Parameters

None

createDFPOnlyBanner

Can be used to add Google demand as backfill for Smart Adserver for instance. It should be called in SmartAdserver *bannerView(_ bannerView: SASBannerView, didFailToLoadWithError error: Error)* callback. Requires the BannerAdUnit to have a GamAdUnitId set using the *setGamAdUnitId* method.

Parameters

containerView: UIView, container View which will be used to display the ad View.

Example Google Ad Server Swift

```
func loadGAMBanner() {
      self.request = DFPRequest()
      let bannerUnit = BannerAdUnit(configId:PBS_CONFIG_ID_300x250, size:
CGSize(width: 300, height: 250))
      dfpBanner = DFPBannerView(adSize:  kGADAdSizeBanner)
      dfpBanner.adUnitID = DFP_BANNER_ADUNIT_ID_320x50
      dfpBanner.rootViewController = self
      dfpBanner.delegate = self # Class should implement the protocol
GADBannerViewDelegate
      dfpBanner.appEventDelegate = self # Class should implement the protocol
GADAppEventDelegate
      appBannerView.addSubview(dfpBanner)
      bannerUnit.fetchDemand(adObject: self.request) { [weak self] (resultCode:
ResultCode) in
      self?.dfpBanner!.load(self?.request)
}


func adView(_ banner: GADBannerView, didReceiveAppEvent name: String, withInfo
info: String?) {
      if (AnalyticsEventType.bidWon.rawValue == name) {
            bannerUnit.sendBidWon(bidWonCacheId: info!)
      }
}


func adView(_ bannerView: GADBannerView) {
      Utils.shared.findPrebidCreativeSize(bannerView,
                                          success: { (size) in
                                              guard let bannerView =
bannerView as? DFPBannerView else {
                                                  return
                                              }
bannerView.resize(GADAdSizeFromCGSize(size))},
                                          failure: { (error) in
                                              print("error: \
(error.localizedDescription)");
        })
}
```

Example Google Ad Server Objective C

```
-(void) loadDFPBanner {
    self.bannerUnit = [[BannerAdUnit alloc] initWithConfigId:@"366c2e80-8932-
4acd-ab9a-a2d7dd5abdfd" size:CGSizeMake(300, 250)];
```

```
    [self.bannerUnit addAdditionalSizeWithSizes: @[[NSValue
valueWithCGSize:CGSizeMake(320, 50)]]];

    [self.bannerUnit setAutoRefreshMillisWithTime:35000];

    self.dfpView = [[DFPBannerView alloc]
initWithAdSize:kGADAdSizeMediumRectangle];

    self.dfpView.rootViewController = self;

    self.dfpView.adUnitID = @"/21807464892/pb_admax_300x250_top";

    self.dfpView.delegate = self;

    self.dfpView.appEventDelegate = self;

    [self.bannerView addSubview:self.dfpView];

    self.request = [[DFPRequest alloc] init];

    [self.bannerUnit fetchDemandWithAdObject:self.request completion:^(enum
ResultCode result) {

        NSLog(@"Prebid demand result %ld", (long)result);

        dispatch_async(dispatch_get_main_queue(), ^{

            [self.dfpView loadRequest:self.request];

        });

    }];

}


-(void) adView:(GADBannerView *)banner didReceiveAppEvent:(NSString *)name
withInfo:(NSString *)info

{

    [self.bannerUnit sendBidWonWithBidWonCacheId:info eventName:name];

}


-(void) adViewDidReceiveAd:(GADBannerView *)bannerView {

    NSLog(@"Ad received");

    [[Utils shared] findPrebidCreativeSize:bannerView

                                   success:^(CGSize size) {

                                       if ([bannerView isKindOfClass:
[DFPBannerView class]]) {

                                           DFPBannerView *dfpBannerView =
(DFPBannerView *)bannerView;

                                           [dfpBannerView
resize:GADAdSizeFromCGSize(size)];

                                       }

                                   } failure:^(NSError * _Nonnull error) {

                                       NSLog(@"error: %@",
error.localizedDescription);

                                   }];

}
```

Example Smart Ad Server with Google Backfill Swift

```
func loadSmartBanner(bannerUnit: AdUnit) {
    let sasAdPlacement: SASAdPlacement = SASAdPlacement(siteId: 305017, pageId:
1109572, formatId: 80250)
    self.sasBanner = SASBannerView(frame: CGRect(x: 0, y: 0, width:
appBannerView.frame.width, height: 50))
    self.sasBanner.autoresizingMask = UIView.AutoresizingMask.flexibleWidth
    self.sasBanner.delegate = self
    self.sasBanner.modalParentViewController = self
    appBannerView.addSubview(sasBanner)

    # For Google Backfill only
    guard let bannerUnit = bannerUnit as? BannerAdUnit else {
            return
        }
    bannerUnit.addAdditionalSize(sizes: [CGSize(width: 320, height: 50)])
    bannerUnit.setGamAdUnitId(gamAdUnitId: "/21807464892/pb_admax_300x250_top")
    bannerUnit.rootViewController = self
    bannerUnit.setGamBannerAdListener(adListener: self)
    # End (For Google Backfill only)

    let admaxBidderAdapter = SASAdmaxBidderAdapter(adUnit: bannerUnit)
    bannerUnit.fetchDemand(adObject: admaxBidderAdapter) { [weak self]
(resultCode: ResultCode) in
        print("Prebid demand fetch for Smart \(resultCode.name())")
        if (resultCode == ResultCode.prebidDemandFetchSuccess) {
            self?.sasBanner!.load(with: sasAdPlacement, bidderAdapter:
admaxBidderAdapter)
        } else {
            self?.sasBanner!.load(with: sasAdPlacement)
        }
    }
}

func bannerViewDidLoad(_ bannerView: SASBannerView) {

    print("SAS bannerViewDidLoad")
}

func bannerView(_ bannerView: SASBannerView, didFailToLoadWithError error:
Error) {
    print("SAS bannerView:didFailToLoadWithError: \
(error.localizedDescription)")
    # For Google Backfill only
    bannerUnit.createDFPOnlyBanner(containerView: appRectangleView)
    # End (For Google Backfill only)
}

# For Google Backfill only

func onAdLoaded(size: GADAdSize) {

    print("DFP Ad loaded with size \(size.size)")

}

func onAdFailedToLoad(error: GADRequestError) {

    print("DFP Ad failed to load with error \(error)")
```

```
}

# End (For Google Backfill only)
```

InterstitialAdUnit Object

Use the InterstitialAdUnit object to create and configure an interstitial ad unit in your app.

Object

InterstitialAdUnit

Create a new Interstitial Ad Unit associated with a banner size and Admax Prebid Server configuration ID that is provided via our interface https://pbsadmin.admaxmedia.io.

Parameters

- *configId*: String, Admax Prebid Server configuration ID

Methods

InterstitialAdUnit inherits all methods from the AdUnit Object.

createDFPOnlyInterstitial

Can be used to add Google demand as backfill for Smart Adserver for instance. It should be called in SmartAdserver *interstitialManager(_ manager: SASInterstitialManager, didFailToLoadWithError error: Error)* callback. Requires the IntertitialAdUnit to have a GamAdUnitId set using the *setGamAdUnitId* method and a rootViewController set using the *rootViewController* property.

Parameters

None

Example Google Ad Server Swift

```
func loadGAMInterstitial() {
      self.request = DFPRequest()
      let interstitialUnit =
InterstitialAdUnit(configId:PBS_CONFIG_ID_INTERSTITIAL)
      dfpInterstitial = DFPInterstitial(adUnitID: DFP_INTERSTITIAL_ADUNIT_ID)
```

```
      dfpInterstitial.delegate = self # Class should implement the protocol
GADInterstitialDelegate

      dfpInterstitial.appEventDelegate = self # Class should implement the
protocol GADAppEventDelegate

      appBannerView.addSubview(dfpBanner)

      interstitialUnit.fetchDemand(adObject: self.request) { (resultCode:
ResultCode) in

      self.dfpInterstitial!.load(self.request)

}


func interstitial(_ interstitial: GADBannerView, didReceiveAppEvent name:
String, withInfo info: String?) {

      if (AnalyticsEventType.bidWon.rawValue == name) {

            interstitial Unit.sendBidWon(bidWonCacheId: info!)

      }

}

func interstitialDidReceiveAd(_ ad: GADInterstitial) {

      if (self.dfpInterstitial?.isReady ?? true) {

            self.dfpInterstitial?.present(fromRootViewController: self)

      }

}
```

Example Google Ad Server Objective C

```
-(void) loadDFPInterstitial {

    self.interstitialUnit = [[InterstitialAdUnit alloc]
initWithConfigId:@"366c2e80-8932-4acd-ab9a-a2d7dd5abdfd"];

    self.dfpInterstitial = [[DFPInterstitial alloc]
initWithAdUnitID:@"/21807464892/pb_admax_interstitial"];

    self.dfpInterstitial.delegate = self;

    self.dfpInterstitial.appEventDelegate = self;

    self.request = [[DFPRequest alloc] init];

    [self.interstitialUnit fetchDemandWithAdObject:self.request
completion:^(enum ResultCode result) {

        NSLog(@"Prebid demand result %ld", (long)result);

        [self.dfpInterstitial loadRequest:self.request];

    }];

}


-(void) interstitial:(GADInterstitial *)interstitial didReceiveAppEvent:
(NSString *)name withInfo:(NSString *)info

{

    [self.interstitialUnit sendBidWonWithBidWonCacheId:info eventName:name];
```

```
}


- (void)interstitialDidReceiveAd:(GADInterstitial *)ad
{
    if (self.dfpInterstitial.isReady)
    {
        NSLog(@"Ad ready");
        [self.dfpInterstitial presentFromRootViewController:self];
    }
    else
    {
        NSLog(@"Ad not ready");
    }
}
```

Example Smart Ad Server with Google backfill Swift

```
func loadSmartInterstitial(adUnit: AdUnit) {
    let sasAdPlacement: SASAdPlacement = SASAdPlacement(siteId: 305017, pageId:
1109572, formatId: 80600)
    sasInterstitial = SASInterstitialManager(placement: sasAdPlacement,
delegate: self)
    # For Google Backfill only
    adUnit.setGamAdUnitId(gamAdUnitId: "/21807464892/pb_admax_interstitial")
    adUnit.rootViewController = self
    # End (For Google Backfill only)
    let admaxBidderAdapter = SASAdmaxBidderAdapter(adUnit: adUnit)
    adUnit.fetchDemand(adObject: admaxBidderAdapter) { [weak self] (resultCode:
ResultCode) in
        print("Prebid demand fetch for Smart \(resultCode.name())")
        if (resultCode == ResultCode.prebidDemandFetchSuccess) {
            self?.sasInterstitial!.load(bidderAdapter: admaxBidderAdapter)
        } else {
            self?.sasInterstitial!.load()
        }
    }
}

func interstitialManager(_ manager: SASInterstitialManager, didLoad ad: SASAd) {
    if (manager == self.sasInterstitial) {
        print("Interstitial ad has been loaded")
        self.sasInterstitial.show(from: self)
    }
}

func interstitialManager(_ manager: SASInterstitialManager,
didFailToLoadWithError error: Error) {
    if (manager == self.sasInterstitial) {
        print("Interstitial ad did fail to load: \(error.localizedDescription)")
        # For Google Backfill only
        self.interstitialUnit.createDfpOnlyInterstitial()
```

```
        # End (For Google Backfill only)
    }
}

func interstitialManager(_ manager: SASInterstitialManager,
didFailToShowWithError error: Error) {
    if (manager == self.sasInterstitial) {
        print("Interstitial ad did fail to show: \(error.localizedDescription)")
    }
}

func interstitialManager(_ manager: SASInterstitialManager, didAppearFrom
viewController: UIViewController) {
    if (manager == self.sasInterstitial) {
        print("Interstitial ad did appear")
    }
}

func interstitialManager(_ manager: SASInterstitialManager, didDisappearFrom
viewController: UIViewController) {
    if (manager == self.sasInterstitial) {
        print("Interstitial ad did disappear")
    }
}
```

GlobalUserTargeting

Year of Birth

You can retrieve and set the year of birth for targeting:

yob = Targeting.shared.yearOfBirth()
Targeting.shared.setYearOfBirth(1989)

Gender

You can retrieve and set the following values for gender:

- female
- male
- unknown

gender = Targeting.shared.gender
Targeting.shared.gender = .female

Global Application Targeting

Store URL

Retrieve and set your app's store URL:

Targeting.shared.itunesID
Targeting.shared.itunesID = itunesID

Global GDPR Targeting

Admax iOS SDK supports the IAB GDPR recommendations (https://github.com/InteractiveAdvertisingBureau/GDPR-Transparency-and-Consent-Framework).

Enable (true) or disable (false) the ability to provide consent.

Targeting.shared.subjectToGDPR = true

Retrieve the consent:

Targeting.shared.subjectToGDPR

Retrieve the consent string:

consentString = Targeting.shared.gdprConsentString

As a publisher, you can set the consent string as follows:

Targeting.shared.gdprConsentString = “consent_string”

Admax iOS SDK also checks if the values are present in the UserDefaults keys specified by the IAB (IAB_GDPR_SubjectToConsent). If the values are also set in these objects, they will be passed in the OpenRTB request object.

Admax iOS SDK API Result Codes

When you use our API to retrieve bids, you will receive a ResultCode indicating whether the request was successful. Here is a description of each code, and where to go to find more information.

Success

- Return Code: SUCCESS
- Description: Admax iOS SDK received at least one valid bid from our Prebid Server and successfully associated Prebid key-values with the appropriate ad server request

Prebid Server Error

- Return Code: PREBID_SERVER_ERROR
- Description: General result code for an unknown error returned from Prebid Server. The actual Prebid Server error message will be exposed to the developer

Invalid accound ID

- Return Code: INVALID_CONFIG_ID
- Description: Admax Prebid Server did not recognize the configuration ID that was passed in on your banner or interstitial ad unit object. Be sure that you have passed in a non-empty configuration ID and that the ID is correct

Invalid size

- Return Code: INVALID_SIZE
- Description: Attempted to add an invalid size to a banner ad unit. This error usually occurs if you have attempted to add multiple sizes on a request to Mopub adserver; Mopub allows only a single size.

Network Error

- Return Code: NETWORK_ERROR
- Description: A network error occurred during the request to our Prebid Server

Timeout

- Return Code: TIME_OUT
- Description: The ad request to Admax Prebid Server exceeded the timeout period

No Bids

- Return Code: NO_BIDS
- Description: Admax Prebid Server responded without returning any valid bids

Empty host URL

- Return Code: INVALID_HOST_URL
- Description: Attempted to define a custom Prebid Server host without providing a host URL