Upgrade Mobile App Performance (MAP) SDK - version 20.32 and above for iOS

Sample Project: https://ac.akamai.com/docs/DOC-75965 | External Link

Prerequisites:

- An iOS app using the MAP SDK with an associated API/license key. If you are trying to integrate the MAP SDK for the first time, please check our article here on how to get started.
- Familiarity with MAP SDK features and configurations. For more information on that, see the MAP SDK Configuration Guide.

To upgrade to MAP 20.32 and above, please follow the instructions in this guide.

How you upgrade the SDK depends on how you added it.

- CocoaPods - If you have added the SDK through CocoaPods, change your podfile to point to the new pod. Change **AkamaiMAP** to **Aka-MAP**. Then run **pod update**.

```
target 'MAP_Unified_OC' do
  use_frameworks!

  pod 'Aka-MAP'

end
```

- If you added the SDK through a ZIP file, first remove the old MAP SDK, then add the latest frameworks manually. Note that two frameworks are now required: AkaMap.framework and AkaCommon.framework.

Deprecated Methods Deprecated methods should be removed and updated with the methods explained below.

License/API key:

MAP license through code - Manual addition of a license was removed. Use your app Plist to add your API key.

```
do {
        let optionsDict = ["license": "license key",
                "segments": [  "segment_1" ,  "segment_2" ],] as [String : Any]
         try self.mapService =   VocServiceFactory.createAkaWebAccelerator(with: self, delegateQueue:
OperationQueue.main, options: optionsDict)
        } catch {
            return false
        }
```

MAP API key through Plist: Update the Plist as follows -
vocsdk becomes **map**
license becomes **api_key**
Update -

▼ com.akamai	Dictionary	(2 items)
▼ map	Dictionary	(2 items)
api_key	String	abcd1234
▼ segments	Array	(1 item)
Item 0	String	images
▼ mpulse	Dictionary	(1 item)
api_key	String	abcd1234

NOTE: You can continue to use the same license and features.

Initialization

Deprecated - `VocServiceFactory.createAkaWebAccelerator(with: self, delegateQueue: OperationQueue.main, options: nil)`
Update - `AkaCommon.configure()`.

Methods and Interface:

Deprecated: `@interface VocServiceFactory : NSObject`
Updated: `@interface AkaMap : NSObject`

Deprecated: `+ (void)setupSessionConfiguration:(nonnull NSURLSessionConfiguration *)sessionConfig;`
Updated: `- (void)interceptSessionsWithConfiguration:(nonnull NSURLSessionConfiguration *)sessionConfig; // now called on [AkaCommon shared]`

Callbacks:

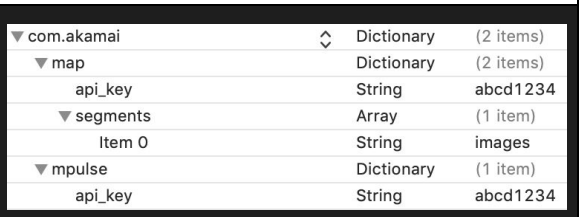
Below Callback methods are deprecated

```
- (void) vocService:(nonnull id<VocService>)vocService didBecomeNotRegistered:(nonnull NSDictionary
*)info;
- (void) vocService:(nonnull id<VocService>)vocService didFailToRegister:(nonnull NSError *)error;
- (void) vocService:(nonnull id<VocService>)vocService didRegister:(nonnull NSDictionary*  )info;
- (void) vocService:(nonnull id<VocService>)vocService didInitialize:(nonnull NSDictionary *)info;
- (void) vocService:(nonnull id<VocService>)vocService didInitialize:(nonnull NSDictionary *)info;
- (void)clearCache:(nullable void (^)(NSError * __nullable))completion AKAMAP_DEPRECATED;
```

Comparison of Legacy vs Unified SDK

In the following steps, the column on the left shows the existing code. Replace this code with the code in the column on the right.

1. License

Legacy SDK	*Unified SDK*
	

2. Header file

You need to add **AkaCommon** and **AkaMap** header file.

Legacy SDK	*Unified SDK*
`#import <VocSdk/VocSdk.h>`	`#import <AkaCommon/AkaCommon.h>` `#import <AkaMap/AkaMap.h>`

3. Initialization

AkaCommon.configure - initializes MAP SDK

Legacy SDK	*Unified SDK*

Swift:
`akaService = VocServiceFactory.createAkaWebAccelerator(with: self, delegateQueue: OperationQueue.main, options: nil)`
 	
Objective-C:
`self.akaService = [VocServiceFactory createAkaWebAcceleratorWithDelegate:self delegateQueue:[NSOperationQueue mainQueue] options:nil error:&error];`
	Swift:
	`AkaCommon.configure()`
	
	Objective-C:
	`[AkaCommon configure];`

4. Debug log

Logging from MAP has not changed. Some additional logs are available via AkaCommon.

Legacy SDK	*Unified SDK*

Swift:
`self.akaService.setDebugConsoleLog(true)`
 	
Objective-C:
`[self.akaService setDebugConsoleLog:true];`
	Swift:
	`self.akaService.setDebugConsoleLog(true)`
	`AkaCommon.shared().debugConsoleEnabled = true`
	
	Objective-C:
	`[self.akaService setDebugConsoleLog:true];`
	`[AkaCommon shared].debugConsoleEnabled = true;`

5. Calling methods

Call MAP methods/API through AkaMap or AkaCommon shared instance:

Swift:

AkaMap.shared()

AkaCommon.shared()

Objective-C:

[AkaMap shared];

[AkaCommon shared]**;**

Use of Interceptor

By default, MAP intercepts traffic going from NSURLSession. Hence, for this scenario you don't need to use a wrapper to intercept traffic going through NSURLSession.

NSURLSession may use either the shared app session or a custom configuration. The SDK automatically accelerates the shared session, so a standard NSURLSession is accelerated by default:

Standard NSURLSession - Objective-C

```
NSURLSession *session = [NSURLSession sharedSession];
NSURL *requestURL = [NSURL URLwithString:@"http://www.akamai.com/"];
[[session dataTaskWithURL:requestURL] resume];
```

Alternatively, an NSURLSession may be created with a custom configuration. The custom configuration must be passed into the SDK for setup. Pass the configuration into the VocServiceFactory call setupSessionConfiguration:

Custom NSURLSession - Objective-C

```
NSURLSessionConfiguration *sessionConfig = [NSURLSessionConfiguration
defaultSessionConfiguration];
// ... modify sessionConfig as required by the app ...
[[AkaCommon shared] interceptSessionsWithConfiguration:sessionConfig];
NSURLSession *session = [NSURLSession sessionWithConfiguration:sessionConfig];

NSURL *requestURL = [NSURL URLwithString :@"http://www.akamai.com/"];
[[session dataTaskWithURL:requestURL] resume];
```

Standard NSURLSession - Swift

```
let url = URL(string: "http://www.akamai.com/")
        let task = URLSession.shared.dataTask(with: url!) {(data, response, error) in
}
task.resume()
```

Custom NSURLSession- Swift

```
let url = URL(string: "http://www.akamai.com/")
let sessionConfig = URLSessionConfiguration.default
AkaCommon.shared().interceptSessions(with: sessionConfig)
let session = URLSession.init(configuration: sessionConfig)
let task = session.dataTask(with: url!) {(data, response, error) in
        }
task.resume()
```

Alamofire - Swift

```
let sessionConfig: URLSessionConfiguration = URLSessionConfiguration.default
AkaCommon.shared().interceptSessions(with: sessionConfig)
let sessionManager = Alamofire.SessionManager(configuration: sessionConfig)
let request = sessionManager.request("url", method: .get).responseJSON { response in
     let _ = sessionManager
     debugPrint(response)
}
```

Moya - Swift

```
static let provider: MoyaProvider<AppApi> = {
let sessionConfig = URLSessionConfiguration.default
AkaCommon.shared().interceptSessions(with: sessionConfig)
```

```
sessionConfig.httpAdditionalHeaders = Manager.defaultHTTPHeaders
let manager = Manager(configuration: sessionConfig)
manager.startRequestsImmediately = false
let provider = MoyaProvider<AppApi>(manager: manager, plugins: [NetworkLoggerPlugin(verbose:
true)])
return provider
}()
```

AFNetworking - Objective-C

```
NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration
defaultSessionConfiguration];
[[AkaCommon shared] interceptSessionsWithConfiguration:sessionConfig];
AFURLSessionManager *manager = [[AFURLSessionManager alloc] initWithSessionConfiguration:
configuration];
NSURL *URL = [NSURL URLWithString:@"url"];
NSURLRequest *request = [NSURLRequest requestWithURL:URL];
NSURLSessionDataTask *dataTask = [manager dataTaskWithRequest:request
completionHandler:^(NSURLResponse *response, id responseObject, NSError *error) {
if (error) {
NSLog(@"Error: %@", error);
} else {
NSLog(@"%@ %@", response, responseObject);
}
}];
[dataTask resume];
```

AFNetworking - Swift

```
let mapSessionConfig: URLSessionConfiguration = URLSessionConfiguration.default
AkaCommon.shared().interceptSessions(with: sessionConfig)
let manager: AFURLSessionManager = AFURLSessionManager.init(sessionConfiguration:
mapSessionConfig)
let uRLRequest = URLRequest(url: URL(string: "url")!)
 let task = manager.dataTask(with: uRLRequest, uploadProgress: { (Progress) in
 }, downloadProgress: { (Progress) in
}, completionHandler: { ( data, response, error) in
     print(response ?? "")
  })
task.resume()
```

Alamofire - Swift

```
let mapSessionConfig: URLSessionConfiguration = URLSessionConfiguration.default
AkaCommon.shared().interceptSessions(with: sessionConfig)
let apiURL = URL(string:  "url")!
let sessionManager = Alamofire.SessionManager(configuration: mapSessionConfig)
_ = sessionManager.request(apiURL, method: .get).responseJSON
 {
```

```
    response in
    let _ = sessionManager// retain
    if let data = response.data, let utf8Text = String(data: data, encoding: .utf8) {
       print("Data: \(utf8Text)")
    }
  }
debugPrint(request)
```

SDWebImage Objective-C

```
NSURLSessionConfiguration *sessionConfig = [NSURLSessionConfiguration
defaultSessionConfiguration];
[[AkaCommon shared] interceptSessionsWithConfiguration:sessionConfig];
[[SDWebImageDownloader sharedDownloader] createNewSessionWithConfiguration:sessionConfig];
[self.imageViewObj sd_setImageWithURL:[NSURL URLWithString:@"url"] placeholderImage:[UIImage
imageNamed:@"placeholder"]];
```

SDWebImage Swift 4.x

```
let mapSessionConfig: URLSessionConfiguration = URLSessionConfiguration.default
AkaCommon.shared().interceptSessions(with: sessionConfig)
SDWebImageDownloader.shared().createNewSession(with: mapSessionConfig)
self.libraryImageView.sd_setImage(with: URL(string: "url"), placeholderImage: UIImage(named:
"placeholder"))
```

SDWebImage Swift 5.x

```
let mapSessionConfig: URLSessionConfiguration = URLSessionConfiguration.default
AkaCommon.shared().interceptSessions(with: sessionConfig)
let sdWebImageDownloaderConfig = SDWebImageDownloaderConfig.default
sdWebImageDownloaderConfig.sessionConfiguration = mapSessionConfig
let downloader = SDWebImageDownloader.init(config: sdWebImageDownloaderConfig)
downloader.downloadImage(with: URL(string: "url")) { (UIImage, Data, Error, Bool) in
      self.libraryImageView.image = UIImage
}
```

PINRemoteImage Swift

```
let mapSessionConfig: URLSessionConfiguration = URLSessionConfiguration.default
AkaCommon.shared().interceptSessions(with: sessionConfig)
self.libraryImageView.pin_setImage(from: URL(string: "url"))
```

Kingfisher Swift

```
let mapSessionConfig: URLSessionConfiguration = URLSessionConfiguration.default
AkaCommon.shared().interceptSessions(with: sessionConfig)
let downloader = ImageDownloader(name: "KFDownloader")
downloader.sessionConfiguration  = mapSessionConfig
```

```
self.libraryImageView.kf.setImage(with: URL(string: "url"), options:
[.downloader(downloader)])
```

Alamofire Image Swift

```
let mapSessionConfig: URLSessionConfiguration = URLSessionConfiguration.default
AkaCommon.shared().interceptSessions(with: sessionConfig)
let imageDownloader = ImageDownloader(
            configuration: mapSessionConfig,    //add MAP session config to the downloader
            downloadPrioritization: .fifo,
            maximumActiveDownloads: 4,
            imageCache: AutoPurgingImageCache()
)
let imageURL = URL(string: "url")!
            let imageURLRequest = URLRequest(url: imageURL)
            imageDownloader.download(imageURLRequest) { response in
                let _ = imageDownloader
                var img = UIImage()
                if response.result.value != nil {
                    img = response.result.value!
                    self.libraryImageView.image = img
                }
            }
```