Upgrade mPulse SDK - version 20.32 and above for iOS

Introduction:

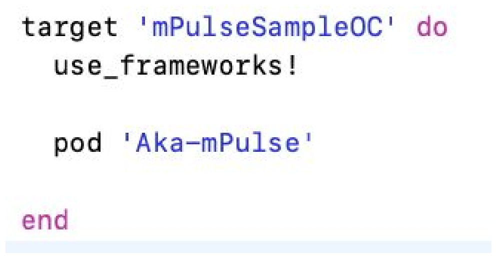
mPulse SDK version 20.32 and above is much easier to integrate and manage. mPulse now also works seamlessly with MAP SDK with smaller combined SDK size. As part of this re-architecture, some of the existing integration approaches have changed. To upgrade mPulse SDK to version 20.32 and above, you have to follow the upgrade steps. This document tells you how to upgrade the MAP SDK.

Sample Project: Download

We've created a sample project that you can use to validate the setup and functionality of the SDK. You can use this sample app to compare against your legacy and make sure everything is working correctly, as outlined in the step below.

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

- Cocoa Pod - If you have added the SDK through Cocoa Pod, then **pod update** will add the latest framework.
  Deprecated: pod 'mPulse'
  Update: pod 'Aka-mPulse'

  ```
  target 'mPulseSampleOC' do
    use_frameworks!

    pod 'Aka-mPulse'

  end
  ```

- If you added the SDK through a ZIP file, you need to remove old SDK of mPulse, and add the latest frameworks manually.

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

License

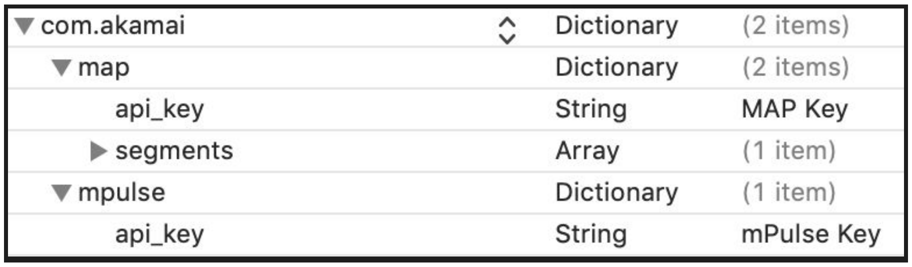
Deprecated: Manual addition of license is deprecated -
Swift:

```
MPulse:mPulse.initialize(withAPIKey: "key")
```

Objective-C:

```
[MPulse initializeWithAPIKey:@"key"];
```

Update: Add the license info in Plist, as shown below.



▼ com.akamai	Dictionary	(2 items)
▼ map	Dictionary	(2 items)
api_key	String	MAP Key
▶ segments	Array	(1 item)
▼ mpulse	Dictionary	(1 item)
api_key	String	mPulse Key

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

Initialization

Swift:

Deprecated: `MPulse.initialize(withAPIKey: "key")`

Update: `AkaCommon.configure()`

`Objective-C:`

Deprecated: `[MPulse initializeWithAPIKey:@"key"];`

Update: `[AkaCommon configure];`

Comparison of Legacy vs Akamai Common Framework

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

Cocoapod:

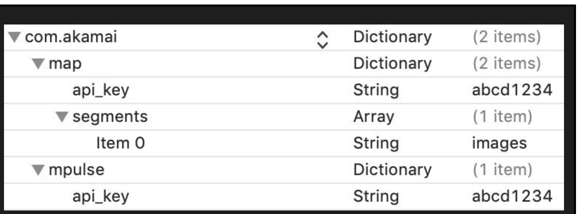
Deprecated: `pod 'mPulse'`

Update: `pod 'Aka-mPulse'`

1. License

Legacy SDK	*Unified SDK*

```
mPulse.initialize(withAPIKey: "key")
```



▼ 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

2. Header file

Add **AkaCommon** and mPulse header file.

Legacy SDK	*Unified SDK*

`#import <MPulse/MPulse.h>`	`#import <AkaCommon/AkaCommon.h>`
	`#import <MPulse/MPulse.h>`

3. Initialization

AkaCommon.configure - initializes mPulse frameworks

Legacy SDK	*Unified SDK*

`Swift:`
`mPulse.initialize(withAPIKey: "key")`
 	
`Objective-C:`
`[MPulse initializeWithAPIKey:@"key"];`
	`Swift:`
	`AkaCommon.configure()`
	
	`Objective-C:`
	`[AkaCommon configure];`

4. Debug log

Use AkaCommon shared instance to call debug method.

Legacy SDK	*Unified SDK*

`Swift:`
`MPulse.setDebug(true)`
 	
`Objective-C:`
`[MPulse setDebug:true];`
	`Swift:`
	`AkaCommon.shared().debugConsoleEnabled =`
	`true`
	
	`Objective-C:`
	`[AkaCommon shared].debugConsoleEnabled =`
	`true;`

5. Calling methods

There is no change in calling mPulse methods, you can call through mPulse shared instance,

Swift:

```
MPulse.sharedInstance()
```

Objective-C:

```
[MPulse sharedInstance]
```

Interceptors:

In the previous version of the mPulse SDK, the SDK automatically intercepted traffic to provide the desired functionality. The latest version uses interceptors which require a wrapper for any custom NSURLSessions or third-party frameworks. The default, shared NSURLSession is automatically intercepted and does not require a wrapper. Hence, for a basic integration using only the shared NSURLSession, you will not need to use a wrapper.

To intercept traffic for custom session NSURLSessions or third-party frameworks, use the **interceptSessions()** method, which sets up NSURLSession configurations to pass requests through the SDK's URL handler. Examples are provided below for custom NSURLSessions and third-party frameworks.

Shared 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 AkaCommon 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];
```

Shared 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 4.x

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

SDWebImage Objective-C 5.x

```
NSURLSessionConfiguration *mapSessionConfig = [NSURLSessionConfiguration
defaultSessionConfiguration];
[[AkaCommon shared] interceptSessionsWithConfiguration:mapSessionConfig];
SDWebImageDownloaderConfig *sdWebImageDownloaderConfig = [SDWebImageDownloaderConfig
defaultDownloaderConfig];
sdWebImageDownloaderConfig.sessionConfiguration = mapSessionConfig;
SDWebImageDownloader *downloader = [[SDWebImageDownloader
alloc]initWithConfig:sdWebImageDownloaderConfig];
[downloader downloadImageWithURL:[NSURL URLWithString:@"url"] completed:^(UIImage *
_Nullable image, NSData * _Nullable data, NSError * _Nullable error, BOOL finished) {
        self.imageView.image = image;
}];
```

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
                }
            }
```