Mobile Accelerator SDK

iOS Integration Guide

February 20, 2019

1 Summary

This document details the process of accelerating Web traffic by integrating the Mobile Application Performance (MAP) SDK with your iOS application.

2 Introduction

The MAP SDK prepositions Web content onto a mobile device based on subscribed content groups ("segments") and policies set up between the client and server. Acceleration and statistics collection are handled internally by the SDK. The MAP SDK API offers configuration options such as defining user subscriptions and control over which connections receive acceleration.

The API also provides developers access to real-time network conditions. This information can be used to augment the user experience by taking necessary actions based on network state.

API calls are available for logging user events to the server. These can be used to associate network traffic originating from the app with events such as tapping a button.

3 Getting Started

The iOS platform provides several ways to request network resources via HTTP and HTTPS. The MAP SDK accelerates both of the direct download approaches, NSURLSession and NSURLConnection. It also enhances Web pages loaded through UIWebView. WKWebView and SFSafariViewController are run by the OS outside of the app process, and are not enhanced by the SDK at this time.

Request Class	Request Type	Requires Extra Setup?
NSURLConnection	Individual file	NO
NSURLSession	Individual file	YES for custom sessions. Shared session is automatic.

UIWebView	Web view	NO

Table 1 - URL Request Types

NSURLConnection and UIWebView are automatically accelerated once the SDK is installed. Each NSURLSession using a custom configuration requires a configuration call. Each approach is covered in the API Reference section.

The MAP SDK library also collects network-related statistics while serving content. These include HTTP time to first byte, request size, response size, duration, and others. These stats are periodically sent to the MAP SDK server for access via the Web portal.

3.1 Requirements and Dependencies

The MAP SDK requires iOS 8 or higher.

A MAP SDK license key is required for registration.

The application's bundle ID (**Project** → choose target → **General** → **Identity** → **Bundle Identifier**) must match the name provided on the MAP Web portal SDK license page. The portal field for this is "iOS Application ID."

In order to preposition content, the app must enable Background Execution and Remote Notifications. See the appendix for guidance.

3.2 SDK Size

The SDK consists of a main "VocSdk" framework and two optional frameworks. These are all fat binaries, meaning they contain compiled code for several architectures. Apple delivers the appropriate architecture for the end user device. Approximate end-user sizes are shown in the arm64 and armv7 columns below.

	Fat size in SDK (MB)	**arm64 size on device (MB)**	**armv7 size on device (MB)**

VocSdk (MAP SDK core)	21.0 (includes bitcode)	**1.8**	**1.6**
mPulse (optional)	141.0 (includes bitcode)	**2.8**	**2.3**
Cronet (optional)	17.0 (no bitcode)	**4.2**	**3.9**
Total download size		**1.8-8.8 depending on optional frameworks**	**1.6-7.8 depending on optional frameworks**

Table 2 - Framework sizes as of MAP SDK 20.1.2

3.3 Installing the iOS SDK

3.3.1 CocoaPods

The MAP SDK for iOS is available as a CocoaPod. CocoaPods is an open source dependency manager for Swift and Objective-C Cocoa projects. Refer to the CocoaPods Getting Started guide if you are unfamiliar with CocoaPods.

1. Once you have a Podfile set up, edit it to add the MAP SDK framework. Here is an example:

```
target 'YOUR_APPLICATION_TARGET_NAME_HERE' do
  use_frameworks!
  pod 'AkamaiMAP'
end
```

   a. Cronet is an optional dependency and it can be included with the MAP SDK framework. Here is an example:

```
target 'YOUR_APPLICATION_TARGET_NAME_HERE' do
  use_frameworks!
  pod 'AkamaiMAP'
  pod 'AkamaiMAP/Cronet'
end
```

2. To install the MAP SDK framework, open terminal, go to your project directory, and run the pod install command:

```
pod install
```

3. Close Xcode, and then open your project .xcworkspace file generated by CocoaPods. From this time onwards, you must use the .xcworkspace file.

3.3.2 Install Manually

1. Download and unzip the MAP SDK zip archive.
2. Add the framework to your Xcode project.
    a. Open your project in Xcode.
    b. Open the File menu.
    c. Click Add Files to <project>.
    d. Choose VocSdk.framework.
    e. Repeat these steps for Cronet.framework if using QUIC.
3. Link the SDK to your project.
    a. Open project settings by clicking the project name in the Project navigator.
    b. Click the General tab.
    c. Under Embedded Binaries, click + and choose VocSdk.framework.
    d. Click Add.
    e. Repeat these steps for Cronet.framework if using it.
4. If using mPulse, then follow the steps below:
    a. Unzip MPulse.framework.
    b. Drag and drop the framework into your Xcode project.
    c. Navigate to the Build Settings section of your target and add the following (if not already present) to the Other Linker Flags setting: -ObjC
    d. Navigate to the Build Phases section of your target and add the following Libraries (if not already present) to the Link Binary With Libraries step:
        i. CoreTelephony.framework

ii. CoreLocation.framework

iii. SystemConfiguration.framework

iv. libc++.dylib or libc++.tdb

v. libz.dylib or libz.tdb

e. If using mPulse from Swift, you must add the following line to the Objective-C bridging header ([project name]-Bridging-Header.h): #import "MPulse/MPulse.h

5. Note: If you are using frameworks with CocoaPods in your project this step is not needed since CocoaPods adds a similar build step to your target.

   VocSdk.framework, mpulse.framework, and Cronet.framework are “fat” frameworks -- i.e. they include code for both simulator and iPhone/iPad devices. Applications submitted to the Apple app store are rejected if they include code for simulator. To remove simulator code from builds for the app store you need to add a build step to your app target. This build step will process only when the build configuration is set to “Release” (e.g., when you archive a build). The script will locate the directory where your executable was built. It will look at each embedded framework and modify it by extracting the supported build architectures for your run (arm7, arm64, etc.) and replacing the SDK framework so that it contains only the slices for those particular architectures.

   a. In your Project Settings, click on Build Phases.

   b. Click the “+” to add a new build phase, and choose “New Run Script Phase.”

   c. Drag this build phase to be the last and make sure it happens after Embed Frameworks build phase.

   d. Optionally, single-click its name and rename it to “strip frameworks”.

   e. Click the arrow to expand the “strip frameworks” row.

   f. Leave the default shell setting of /bin/sh.

   g. Add the following text to the script area. The strip_frameworks.sh path is relative to your .xcodeproj file and may need modification depending on where you unzipped it (map_sdk in this example).

```
./map_sdk/strip_frameworks.sh
```

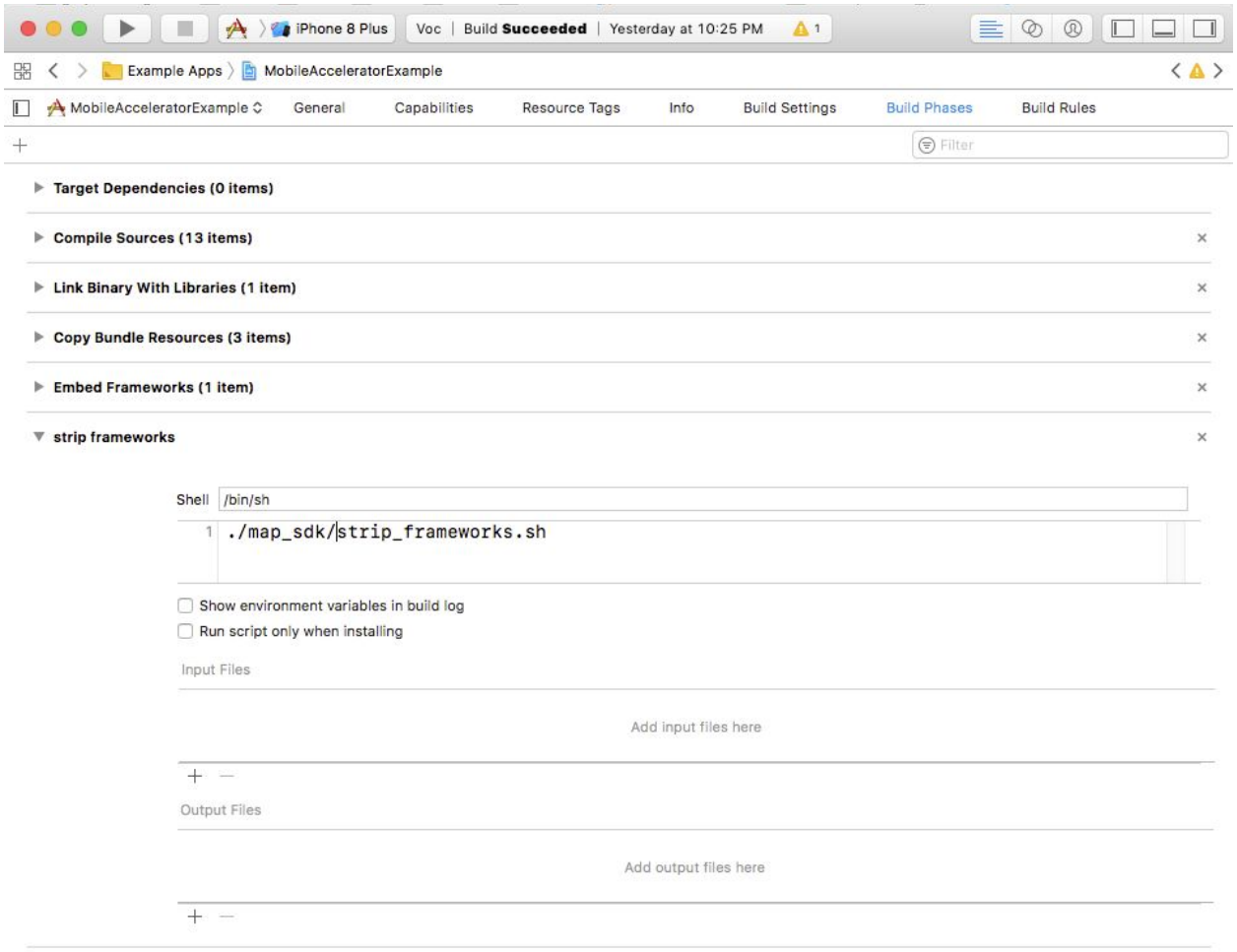
In this example, the folder structure is as follows:

```
/myproject/myproject.xcodeproj

/myproject/map_sdk/strip_frameworks.sh

/myproject/map_sdk/VocSdk.framework
```

The final build phase should look like this:



4 Integrating with your iOS Application

4.1 Including the SDK

The SDK is made available to your code through the AkaWebAccelerator protocol. Import the VocSdk header in any class files where the SDK is required. Also define a single AkaWebAccelerator object for your app. Create your VocService object in the app delegate to make the SDK available early in the app lifecycle and to simplify access to the VocService from other classes.

```
#import <VocSdk/VocSdk.h>

@property (strong, nonatomic) id<AkaWebAccelerator> akaService;
```

4.2 Initialization

The SDK is initialized by the VocServiceFactory call createAkaWebAccelerationWithDelegate:delegateQueue:options:error:. Initialization and registration should take place at startup to ensure that acceleration is available as early as possible. The recommended place for this is in AppDelegate's application:didFinishLaunchingWithOptions:. The create call inputs a reference to the SDK delegate (see SDK Delegate) as well as a configuration options dictionary. The SDK delegate is the class you designate to respond to SDK activity.

Registration requires a valid SDK license and an array of segment names for which to download content. Each segment is a unique string representing a particular content set such as "daily deals," and is created during the content ingest phase. The name may be any string that does not reveal personal information. Segments passed during the register call are subscribed on initial registration; this saves the step of subscribing later. The array may be empty. Further subscriptions may be added or removed later via the subscribeSegments call.

It is strongly recommended that PII (Personally Identifiable Information) not be directly used in naming your content segments.

During initialization, the SDK will attempt to register using a license key and an optional array of segments, if found in one of three places:

1. [**preferred approach**] The Info.plist for the app, using the key “com.akamai.vocsdk.” If found, the dictionary pointed to by this entry is parsed for the “license” key and “segments” and, if found, the values associated with those dictionary entries will be used as the license key and segments for registration purposes:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>CFBundleDevelopmentRegion</key>
        <string>en</string>
        <key>CFBundleDisplayName</key>
        <string>MAPSdkExample</string>

        <!-- ... other Info.plist keys omitted ... -->

        <key>com.akamai</key>
        <dict>
            <key>vocsdk</key>
            <dict>
                <key>license</key>
                <string>your_license_key_goes_here</string>
                <key>segments</key>
                <array>
                <string>segment_1</string>
                <string>segment_2</string>
                </array>
```

```
            </dict>
        </dict>
</dict>
</plist>
```

2. [**option 2**] A configuration file for the app, identified by the value in the Info.plist key "com.akamai.vocsdk.config.file." The value is a path to a config file, relative to the main bundle for your app. This file may be a property list (similar to the Info.plist for your app):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>license</key>
        <string>your_license_key_goes_here</string>
        <key>segments</key>
        <array>
        <string>segment_1</string>
        <string>segment_2</string>
        </array>
</dict>
</plist>
```

Alternatively, the config file may be a JSON file, as shown in the example below:

```
{
   "license" : "your_license_key_goes_here",
   "segments" : [
       "segment_1",
```

```
        "Segment_2"
    ]
}
```

3. [**option 3**] The options dictionary passed to the createServiceWithDelegate:delegateQueue:options:error: method, using the dictionary keys “license” and “segments”:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    NSError *error = nil;
   NSDictionary *options = @{
        @"license" : @"your_license_key_goes_here",
        @"segments" : @[ @"segment_1", @"segment_2" ],
  };
   self.akaService = [VocServiceFactory createAkaWebAcceleratorWithDelegate:self
                      delegateQueue:[NSOperationQueue mainQueue]
                              options:options
                              error:&error];
    if (!self.akaService) {
        // error handling - could not start service
        return NO;
    }
   // app initialized
    return YES;
}
```

Options passed in the options parameter dictionary take precedence over Info.plist options and file options in the following order of precedence:

1. The options dictionary
2. The Info.plist

3. Configuration property list file or JSON file in bundle

4.3 Registration

Registration is a one-time event that activates the MAP SDK service with a particular license key and an array of segments. The key is linked to prepositioned content, generation of usage statistics, and other SDK capabilities. It is required for most SDK features so registration occurs early within the application.

The service object returned from initialization (see Initialization) has a state property that indicates whether registration already succeeded on this device. If vocService.state is equal to VOCServiceStateNotRegistered then registration may have failed, or may not have been attempted for some reason (e.g., there is no network connectivity). The SDK automatically attempts to re-register on unsuccessful registration due to device failures and server errors.

Once registration is successful, the SDK calls VocServiceDelegate's didRegister:. This can be used to report or log any problems starting the SDK. This callback is made every time the app starts. The first time the app registers, it is called in response to registerWithLicense. After that, the app is already registered so didRegister: is called in response to createServiceWithDelegate.

The service delegate also receives didInitialize: to indicate that all services are actively running.

4.4 Updating Segment Subscriptions

The list of subscribed segments may be changed any time after registration. Pass a set of segment names to the SDK service call subscribeSegments. This immediately initiates a check for changes to subscribed content. Unsubscribed files are purged from cache, unchanged files remain in cache, and new files are queued to begin downloading.

```
NSSet *segments = [NSSet setWithArray:@[@"basics", @"daily-deals"]];
[appDelegate.akaService subscribeSegments:segments];
```

To unsubscribe from a segment, pass the whole list of subscribed segments excluding the one(s) you want to remove. An empty set is used to unsubscribe from all segments.

For example, if you are subscribed to segments A, B, and C and you want to remove segment B, call subscribeSegments with A and C. Since B is no longer subscribed, its contents will be purged. Files are kept in cache if they remain in at least one subscribed segment.

4.5 SDK Delegate

The SDK notifies your app of various events throughout its life cycle. Messages are sent asynchronously to an SDK delegate in your code that implements the VocServiceDelegate protocol. All of the delegate methods are optional.

```
- (void) vocService:(nonnull VocService *)vocService didBecomeNotRegistered:(nonnull
NSDictionary *)info;
- (void) vocService:(nonnull VocService *)vocService didFailToRegister:(nonnull NSError
*)error;
- (void) vocService:(nonnull VocService *)vocService didRegister:(nonnull NSDictionary
*)info;
- (void) vocService:(nonnull VocService *)vocService didInitialize:(nonnull NSDictionary
*)info;
- (void) vocService:(nonnull VocService *)vocService
       didReceiveChallengeForRequest:(nonnull NSURLRequest *)originalRequest
       currentRequest:(nonnull NSURLRequest *)currentRequest
       challenge:(nonnull NSURLAuthenticationChallenge *)challenge
       modifiedTrust:(nullable SecTrustRef) modifiedTrust
       completion:(nonnull void (^)(NSURLSessionAuthChallengeDisposition disposition,
              NSURLCredential * _Nullable credential))completion;
- (void) vocService:(nonnull VocService *)vocService itemsDiscovered:(nonnull NSArray
*)items;
- (void) vocService:(nonnull VocService *)vocService itemsStartDownloading:(nonnull NSArray
*)items;
- (void) vocService:(nonnull VocService *)vocService itemsDownloaded:(nonnull NSArray
*)items;
- (void) vocService:(nonnull VocService *)vocService itemsEvicted:(nonnull NSArray *)items;
```

The SDK delegate is set in the SDK initialization call. Typically, this is the app delegate since its lifetime will span that of the SDK, from registration until shutdown. Define your app delegate as follows to implement the SDK delegate protocol.

```
@interface AppDelegate : UIResponder <UIApplicationDelegate, VocServiceDelegate>
```

5 API Reference

5.1 Prepositioned Content

Prepositioned content begins loading onto the device as soon as the user registers with user segments, or registers and later joins user segments. This happens automatically while your program runs. Network requests are served from matching prepositioned content. If the content is not prepositioned then it will be fetched from the network. Your app can listen for the AkaService -didInitialize: callback to know when the SDK has begun handling requests. This is a one-time event that happens after creating the service.

5.2 NSURLConnection

Requests using NSURLConnection will take advantage of preloaded content without any modifications. For example, an asynchronous NSURLConnection can be created as before and will see the benefits of the SDK's acceleration.

```
NSURL *requestURL = [NSURL URLWithString:@"https://www.akamai.com"];
NSURLRequest *request = [NSURLRequest requestWithURL:requestURL];
NSURLConnection *connection = [[NSURLConnection alloc] initWithRequest:request
delegate:self];
// Followed by the asynchronous response handlers: connection:didReceiveResponse:,
connection:didReceiveData:, etc.
```

Synchronous connections are similarly straightforward. No changes to the connection are required to benefit from MAP SDK acceleration.

```
NSData *data = [NSURLConnection sendSynchronousRequest:request returningResponse:&response
error:&error];
```

5.3 UIWebView

UIWebView will also use prepositioned content automatically and without modification.

5.4 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:

```
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:.

```
NSURLSessionConfiguration *sessionConfig = [NSURLSessionConfiguration
defaultSessionConfiguration];
// ... modify sessionConfig as required by the app ...
[VocServiceFactory setupSessionConfiguration:sessionConfig]; // sessionConfig now uses SDK
acceleration
NSURLSession *session = [NSURLSession sessionWithConfiguration:sessionConfig delegate:self
delegateQueue:nil];
NSURL *requestURL = [NSURL URLwithString:@"http://www.akamai.com/"];
[[session dataTaskWithURL:requestURL] resume];
```

5.5 Custom Event Tracking

Custom events are actions triggered by the user such as tapping a button or opening a particular screen. These are defined by the developer. The SDK provides an API for logging them to the server and optionally timing them.

Custom events are classified as timed or instantaneous. A timed event has associated start and end points. The two endpoints are paired by calling startEvent: and stopEvent: with matching event names, and the time between these endpoints is recorded. In addition to logging durations, timed events are useful for monitoring the network activity between endpoints. For example, custom event starting and stopping points can be recorded in-line with network activity and then reviewed from the Web portal.

Note that unrelated, asynchronous requests may be recorded during user events depending on your app design.

If mPulse is enabled, MAP SDK will implicitly set an mPulse Page View Group when startEvent("<Name>") is called. The client can reset the mPulse Page View Group by calling stopEvent("<Name>").

Also, when startEvent and stopEvent identified with "<Name>" is called, a custom timer with name "<Name>" and the same interval will be triggered. The custom timer has to be defined in the mPulse portal.

```
[akaService startEvent:@"Event name"];
// activity
[akaService stopEvent:@"Event name"];
```

Instantaneous events are recorded in the server logs along with the time they were executed. They are useful for recording a sequence of activities or to form a timeline of events. Note that instantaneous events are in Tech Preview and are not yet displayed in the portal.

```
// instantaneous event
[akaService logEvent:@"tapped home button"];
```

MAP SDK can identify mPulse Custom Metrics based on the URL patterns defined by the client and report this to the mPulse portal. The client needs to configure the Custom Metric in the mPulse portal and then define the URL patterns associated with the metric in info.plist file (refer section 4.2 for more details about this file). An example URL pattern definition is as below. Please ensure not to add any data to the event name that has privacy implications.

```
        <!-- ... other Info.plist header omitted ... -->

        <key>com.akamai</key>
        <dict>
            <key>vocsdk</key>
            <dict>
                <!-- ... other Info.plist keys omitted ... -->
                <key>custom_metric</key>
                <dict>
                  <key>ShoppingCart</key>
                  <array>
                    <string>https://www.companyA.com/cart/</string>
                    <string>https://www.companyB.com/cart/</string>
                  </array>
                </dict>
                  <key>Checkout</key>
                  <array>
                    <string>https://www.companyA.com/checkout/</string>
                    <string>https://www.companyB.com/checkout/</string>
                  </array>
                </dict>
            </dict>
        </dict>
        <!-- ... other Info.plist footer omitted ... -->
```

5.6 Network Aware Experience

The SDK provides API access to the client-side network quality state in order to help developers augment client requests. The return value is either excellent, good, or poor. The meaning of these values is defined in the configuration portal.

The next example displays the latest network status and suggests how loading a Web site may be tweaked as a result.

```
id<VocNetworkQuality> networkQuality = appDelegate().akaService.networkQuality;
switch([networkQuality qualityStatus]) {
        case VocNetworkQualityPoor:
            [self flashMessage:nil withTitle:@"Network Quality: Poor"];
            // Exit download
            break;
        case VocNetworkQualityGood:
            [self flashMessage:nil withTitle:@"Network Quality: Good"];
```

```
            // Throttle download
            break;
        case VocNetworkQualityExcellent:
            [self flashMessage:nil withTitle:@"Network Quality: Excellent"];
            // Download content
            break;
        case VocNetworkQualityUnknown:
            [self flashMessage:nil withTitle:@"Network Quality: Unknown"];
            break;
}
```

5.7 Cache-Control Request Parameters

Content is transparently served from either the server or the cache. The SDK ensures delivery of fresh content by following content expiration headers, performing refreshes as necessary. In certain cases, it may be desirable to override this behavior. For example, in case of poor connectivity, a caller may be okay using stale responses for a particular request. In another case, a caller may decide to force cached content to be revalidated by the origin server by controlling its expiry time and date.

These are standard HTTP parameters and may be added directly to the NSURLRequest:

```
NSMutableDictionary *cacheControlHeaders = [NSMutableDictionary new];
cacheControlHeaders[@"Cache-Control"] = @"no-cache";

NSURL *requestURL = [NSURL URLWithString:@"https://www.akamai.com/some_image.jpg"];
NSMutableURLRequest *mRequest = [NSMutableURLRequest requestWithURL:requestURL];
[mRequest setAllHTTPHeaderFields:cacheControlHeaders];

NSURLSessionDataTask *dataTask = [self.mySession dataTaskWithRequest: mRequest];
[dataTask resume];
```

SDK cache behavior can be controlled with the following key-value pairs.

- Pragma:no-cache: Forces SDK to revalidate cached response.
- Cache-Control:no-cache: Same as Pragma:no-cache.
- Cache-Control:max-age='x': Forces SDK to select expiry for the content as Min('x', expiry calculated from response headers).

- Cache-Control:max-stale='x': If assigned a value, the client is willing to accept a response that has exceeded its expiration time by no more than the specified number of seconds. If present and no value is assigned to max-stale, then the client is willing to accept a stale response of any age. Developers can use this under poor network conditions to serve stale responses.

Note: max-age/max-stale is ignored if no-cache is present.

5.8 Debugging Options

SDK error messages are output to the Xcode console. Developers may print extended debug output to the console with the following calls. -setDebugConsoleLog: enables real-time extended information, while the other two calls, -printManifest and -printCurrentCapabilities, issue once-per-use information.

```
// enable real-time extended debug info to Xcode console
[self.akaService setDebugConsoleLog:YES];

// print subscribed segments, followed by each URL with its download status
[self.akaService printManifest];

// print last received SDK capabilities as configured through the portal
[self.akaService printCurrentCapabilities];
```

The -debugSendAnalytics call may be used to test that records are sent correctly from your app. It immediately sends the latest batch of analytics from the device and reports to the developer console. *This results in additional uploads and should* not *be used in production code.*

```
// Debug only - test analytics upload by sending outside of regular cycle
[self.akaService debugSendAnalytics];
```

Note: there will be a delay before analytics are aggregated for display on the portal.

5.9 Custom TLS Certificate Handling

MAP SDK uses the device's default certificate chain to decide which servers to trust. There are cases where an app needs to customize this behavior. These apps should implement the optional delegate callback:

```
- (void) vocService:(nonnull VocService *)vocService
       didReceiveChallengeForRequest:(nonnull NSURLRequest *)originalRequest
       currentRequest:(nonnull NSURLRequest *)currentRequest
       challenge:(nonnull NSURLAuthenticationChallenge *)challenge
       modifiedTrust:(nullable SecTrustRef) modifiedTrust
       completion:(nonnull void (^)(NSURLSessionAuthChallengeDisposition disposition,
              NSURLCredential * _Nullable credential))completion;
```

This passes the app several pieces of information, with full details in the header file:

- the original request made by the app for identifying the URL in question
- the TLS server challenge
- the modified trust object that can be used for verification
- a completion block to be called with the result of the evaluation

The callback will be made for all requests and prepositioned downloads made through the SDK. Its usage and parameters are fully explained in the header file.

6 Appendix - Requirements and Dependencies

6.1 Background Execution

The MAP SDK downloads content while the application is running. Various factors determine when to start downloading, how much to download, and when to pause downloads. Influencing factors include the state of the mobile network and the quality state of the provider network.

When your app is in the foreground, downloads are happening without any need for changes to your code. For best results, the MAP SDK should also be able to download with your app in background. There are two background execution modes that MAP SDK uses to download content. Enable both of these modes from the Xcode target settings → Capabilities tab → Background Modes section:

- Remote notifications (remote-notification)

- Background fetch (fetch)

To enable background fetch in MAP SDK, you need to implement the system method

UIApplicationDelegate application:performFetchWithCompletionHandler:

and, from there, pass the message to the AkaService by calling

AkaService application:performFetchWithCompletionHandler:

Here is what that looks like:

```
- (void)application:(UIApplication *)application performFetchWithCompletionHandler:(void
(^)(UIBackgroundFetchResult result))completionHandler
{
    if (![self.akaService application:application
performFetchWithCompletionHandler:completionHandler]) {
        completionHandler(UIBackgroundFetchResultNoData);
    }
}
```

6.2 Remote Notifications

To make remote notifications work in the MAP SDK you need to 1) enable the SDK backend to send push notifications to your app, and 2) make the supporting code changes.

To enable the MAP SDK backend to send push notifications to your app, you need to upload your app push certificate (APNS) to the MAP license management portal. The MAP backend must have a valid APNS certificate for your app at all times otherwise push notifications will not work. If you revoke or renew your certificate, make sure you upload it to the MAP license management portal. Instructions on how to generate an APNS push certificate are available on Apple's web site at: https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/AddingCapabilities/AddingCapabilities.html#//apple_ref/doc/uid/TP40012582-CH26-SW11

The first step for your app is to hand the push token to the SDK in the AppDelegate call application:didRegisterForRemoteNotificationsWithDeviceToken:.

```
- (void)application:(UIApplication*)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)deviceToken
{
    [self.akaService setDevicePushToken:deviceToken];
}
```

Also in your application delegate, implement the system method

UIApplicationDelegate application:didReceiveRemoteNotification:fetchCompletionHandler:

and pass this notification to the AkaService

AkaService application:didReceiveRemoteNotification:fetchCompletionHandler:

For example:

```
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult
result))completionHandler
{
    if ([self.akaService application:application didReceiveRemoteNotification:userInfo
fetchCompletionHandler:completionHandler]) {
        // remote notification was for MAP SDK
        return;
    }

    // remote notification is not for MAP SDK, handle remote notification
    completionHandler(UIBackgroundFetchResultNoData);
}
```

6.3 Bitcode

The SDK is compiled with bitcode enabled so it will work in both bitcode- and non-bitcode apps.

6.4 App Transport Security

Starting in iOS 9.0, a new app security feature called App Transport Security (ATS) has been introduced by Apple and it is enabled by default. With ATS enabled, connections must use secure HTTPS instead of HTTP. Additionally, if the app contents that MAP SDK needs to download contain non-SSL items, those downloads will fail. Application developers must ensure that either

1. [preferred] HTTPS is used for all content URLs, or
2. [insecure] ATS exceptions can be added for certain domains by adding the following key-subkey to the app's info.plist file

Optional ATS key to allow insecure (HTTP) content:

```
<key>NSAppTransportSecurity</key>
<dict>
        <key>NSExceptionDomains</key>
        <dict>
                <key>your-domain.com</key>
                <dict>
                        <key>NSIncludesSubdomains</key>
                        <true/>
                        <key>NSExceptionAllowsInsecureHTTPLoads</key>
                        <true/>
                </dict>
                <key>your-other-domain.com</key>
                <dict>
                        <key>NSIncludesSubdomains</key>
                        <true/>
                        <key>NSExceptionAllowsInsecureHTTPLoads</key>
                        <true/>
                </dict>
        </dict>
  </dict>
```

6.5 Disabling Redirect Behavior

MAP SDK automatically follows redirects. To instead return the redirect response to your app, set the following MAP config property.

```
id<AkaWebAccelerator> akaService;
akaService.config.autoFollowRedirects = NO;
```

Your app will then receive -didReceiveResponse: with the status code 301, 302, etc., and a location header to which you can choose to create a new request.

This property supersedes the -willPerformHTTPRedirection: call, which should be omitted or return the recommended request. Do not return nil even if you choose to not auto-redirect. MAP SDK will follow or not follow based on the *autoFollowRedirects* property.