Mobile App Performance SDK

Version 21.60
September 17th, 2020
In App Size: 269.1 KB

Android Integration Guide

1 Summary

This document details the process of integrating MAP SDK with your Android application to accelerate web traffic.

2 Introduction

The SDK internally takes care of pre-positioning the content based on user preferences and policies set up between client and server. SDK provides APIs (networking libraries) to be used by developers that take care of acceleration and stats collection.

The SDK provides an API for developers to access real time network conditions such as congestion state. This information can be used to augment user experience by taking necessary action based on network state.

In addition, SDK also provides APIs for logging user events which could be used to associate traffic originated from the app with events such as the click of a button.

3 Getting Started

Network Interception

- Network requests will be automatically instrumented as long as they are sent from java.net.URLConnection.
- Another way to access content over the network in an app is through WebViews. SDK provides an API for developers which delegates all network-related requests originating from WebViews to the *MAP SDK.*
- If network requests are made through OkHttpClient or the library that uses it (examples are Picassa, Retrofit etc.), the OkHttpClient needs to be configured to use the MAP interceptor.

The *SDK* also collects network-related statistics (such as HTTP time to first byte, request size, response size, duration etc.) alongside serving content. These stats are sent periodically to a server and can be later accessed via portal.

3.1 Requirements and Dependencies

The SDK supports API 15 and above (minSdkVersion). The target/compile SDK should be atleast set to API 28 and it is also required by Google for any app updates from November 2019. Google no longer maintains support libraries up until version 28 and has made androidx the default support library in Android Studio. MAP SDK has migrated from Android Support libraries to androidx-packaged library artifacts. This will not only make our SDK better, but also allows us to use the latest Jetpack features. If you have not migrated to androidx library artifacts, add the following two lines to your gradle.properties file to use MAP SDK.

```
android.useAndroidX=true
android.enableJetifier=true
```

Androidx also requires minimum build tools version to be set at **'com.android.tools.build:gradle:3.2.0'** in the root *build.gradle* file. The minimum gradle version should be set at **gradle-4.6** in the *gradle-wrapper.properties*. MAP SDK has a dependency on Work Manager (**'androidx.work:work-runtime:2.4.0')** and the dependency is automatically pulled into your project.

3.2 Installing the Android SDK

In Android Studio, open the build.gradle file in the **app** directory (*not the one in the root folder*) and edit the dependencies sub-section to include aka-map. Use the latest version of aka-map from jcenter.

```
dependencies {
       implementation 'com.akamai.android:aka-map:21.60.+'
}
```

The project gradle file should have jcenter() by default. If not, add it as shown below.

```
allprojects {
   repositories {
       jcenter()
    }
}
```

The SDK requires following permissions for full functionality:

```
<manifest . . . >
     <uses-permission android:name="android.permission.INTERNET" />
     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
     <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
</manifest>
```

4 Integration with your Android Application

4.1 Initialization

MAP SDK is initialized automatically and the client does not have to explicitly initialize the SDK. After initialization, the SDK needs to be registered with an API key.

Enable the SDK

Client AndroidManifest.xml will need to be updated in order to complete the SDK integration.

```
<application
....

<!-- Refers to sdk init file in res/xml/ -->
<meta-data
android:name="com.akamai.android.sdk"
android:resource="@xml/akamai_sdk_init" />


....
</application>
```

The SDK uses an xml file to look up the license key to authorize the client app. This information is stored in a file android_sdk_init.xml in the client app's resources folder. The sample file is shown below.

Segments are associated with MAP SDK's preposition feature.Providing the segments in the XML file will be considered as a subscription request. Any content associated to the subscribed segments automatically starts getting prepositioned based on network policy(wifi/cellular) defined.

In …./main/res/xml (create if it doesn't exist!) folder, add a new file android_sdk_init.xml.

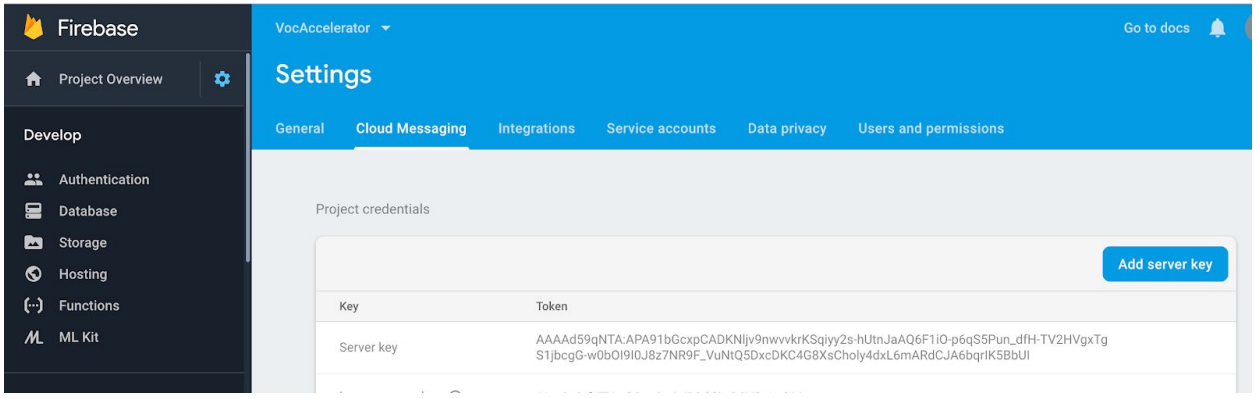
```
<?xml version="1.0" encoding="utf-8"?>
<com_akamai_sdk_init>
<!-- SDK license key created on portal→
<com_akamai_map_license_key></com_akamai_map_license_key>

<!-- In case of MAP license, this is comma separated list of segments to register with
(optional)-->

<com_akamai_map_segments>small,medium</com_akamai_map_segments>
 </com_akamai_sdk_init>
```

4.2 Integrating with Firebase Cloud Messaging (Optional)

If the app does not have a need for prepositioning content in background then this step is optional. MAP SDK uses Firebase Cloud notifications to mainly sync prepositioning content and sdk config in background,

In order for FCM background notifications to work - follow the firebase messaging integration guide as documented at Google Firebase website(https://console.firebase.google.com) . After firebase is integrated in the app, complete the following two steps:

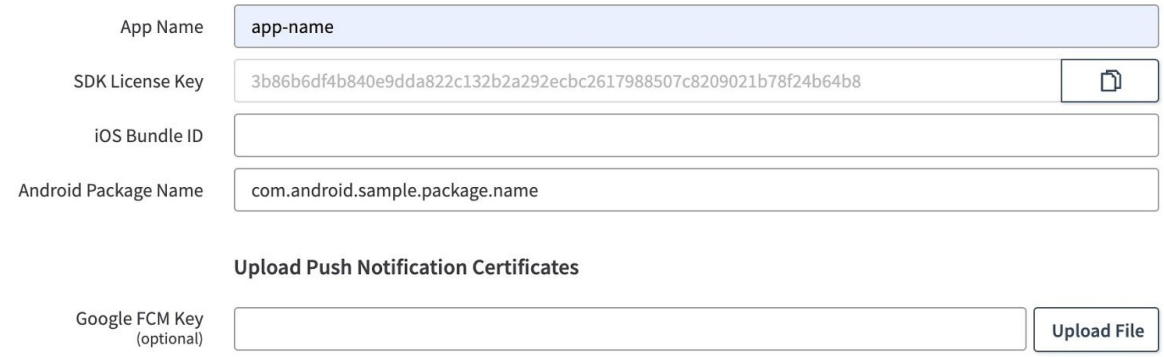
1. Add/Update the server api key to the portal for the required app. The server api key could be found on the firebase console project settings as shown below.



Then add/update the key to MAP SDK license portal in the “Google FCM Key” field on MAP Portal as shown below. The API Key is used by MAP Control Server to trigger map related notifications - for prepositioning any content configured for the app .

While you set up your app on the Firebase website, the portal already has provided you with step by step instructions on how to integrate your app with Firebase. If you have missed the instructions, please follow - https://firebase.google.com/docs/cloud-messaging/android/client

MAP SDK supports firebase-messaging from version 17.3.4 and above.



2. The app needs to provide hooks in its implementation of its FirebaseMessagingService to pass the messages to SDK. A returned ‘true’ value means the message was meant for the MAP SDK and the app doesn’t need to do any further processing on it. A ‘false’ is returned if it is not a MAP SDK message. An example is included below:

```
import com.akamai.android.sdk.AkaCommon;
...
public class MyFirebaseMessagingService extends FirebaseMessagingService {

    @Override
    public void onMessageReceived(RemoteMessage message) {
        // If handleFirebaseMessage returns true, the message is for map sdk.
        if (AkaCommon.getInstance().handlePushNotification(message.getData())) {
            return;
        }

        // Handling for app messages.
        ...
    }

    @Override
    public void onNewToken(String newToken) {

        // App handling for tokens.
        ...
    }
}
```

4.3 Updating segment subscription

The list of subscribed segments may be changed any time after initialization. Pass an array of segment names using AkaMap::subscribeSegments().

```
import com.akamai.android.sdk.AkaMap;

/**
* API method to update segment subscription once registered. Multiple calls to this API
would update previous subscription.

* @param segments Set of segment names. Passing an empty set would clear all segment
*                 subscriptions including current ones.
* Usage:
* AkaMap.getInstance().subscribeSegments(segments);
*/
public boolean subscribeSegments(@NonNull final Set<String> segments)
```

5 API Reference

In this section, we see examples of how to use APIs provided by the SDK.

5.1 Using SDK with Android HttpsUrlConnection/HttpUrlConnection

By default, after successful initialization SDK intercepts all the HttpUrlConnection and HttpsUrlConnection requests. There is no additional code change required by the application. SDK intercepts these requests by setting a global URLStreamHandlerFactory. All the relevant network statistics related to each request will be captured by the SDK.

5.2 Using SDK with WebViews

In order to accelerate traffic originated from WebViews, the SDK provides the following custom WebViewClient.
AkaWebViewL21Client - For Android API level 21 and above.
AkaWebViewL15Client - For Android API level 15 and above.

Both WebViewClients delegate all network calls via the MAP SDK library internally. The only difference between the two is that the L21 WebViewClient uses newer APIs added in level 21 and above.
The caller can use the appropriate client depending on API level as follows:

```
import com.akamai.android.sdk.net.webkit.AkaWebViewL15Client;
import com.akamai.android.sdk.net.webkit.AkaWebViewL21Client;
...

WebView webView;
...
if (Build.VERSION.SDK_INT >= 21) {
        webView.setWebViewClient(new AkaWebViewL21Client());
} else {
        webView.setWebViewClient(new AkaWebViewL15Client());
}
```

Note - Currently, both WebViewClients do not handle POST requests as there is no API provided by Android platform to access POST data.

5.3 Using SDK with Third party HTTP client wrappers

In order to accelerate traffic originating from Third party http clients like Okhttp, Retrofit, Picasso, an interceptor needs to be added to the request. Sample interceptor classes for all these libraries have been added in the **wrappers folder under MAP** in the Akamai_Android_SDKs.zip file.

OkHttp

An interceptor for OkHttp needs to be added to the OkHttpClient.Builder as below

```
OkHttpClient client = new OkHttpClient.Builder()
        .addInterceptor(new AkaOkHttpInterceptor())
        .build();
Request request = new Request.Builder()
        .url(uri)
        .build();
Response response = client.newCall(request).execute();
```

Retrofit - Version 1

An interceptor for Retrofit needs to be added to the RestAdapter class as below

```
RestAdapter.Builder builder =
        new RestAdapter.Builder()
                .setEndpoint(MapUrlTestMockedServer.HTTPSHOST)
                .setClient(new AkaRetrofitClient(getContext()));
RestAdapter adapter = builder.build();
RetrofitApi api = adapter.create(RetrofitApi.class);
```

Retrofit - Version 2

An interceptor for Retrofit 2 needs to be added as the client to retrofit as shown below

```
Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(uri)
        .addConverterFactory(GsonConverterFactory.create())
        .client(AkaRetrofit2Client.getClient())
        .build();
Retrofit2Api client =  retrofit.create(Retrofit2Api.class);
```

```
Call<ResponseBody> call = client.get(path);
```

Picasso - version 2.71828 and above

A downloader for Picasso needs to be added as below.

```
Picasso picasso = new Picasso.Builder(getContext())
       .downloader(new OkHttp3Downloader(AkaPicassoDownloader.getClient()))
       .build();
Picasso.setSingletonInstance(picasso);
```

Picasso - version 2.5.2 and below

```
Picasso picasso = new Picasso.Builder(getContext())
       .downloader(new AkaPicassoDownloader_upto_v2_5_2(getContext()))
       .build();
Picasso.setSingletonInstance(picasso);
```

Glide

A downloader for Glide needs to be added as below.
Sample AppGlide module:

```
@GlideModule
@Excludes(OkHttpLibraryGlideModule.class)
public class AppGlide extends AppGlideModule {

   @Override
   public void registerComponents(@NonNull Context context, @NonNull Glide glide, @NonNull
Registry registry) {
       super.registerComponents(context, glide, registry);
       ...
       registry.replace(GlideUrl.class, InputStream.class, new
OkHttpUrlLoader.Factory(AkaGlideClient.getClient()));
   }
}
```

Volley

For volley, an interceptor for OkHttp needs to be added in OkHttpStack class which extends BaseHttpStack, then it should be added to the OkHttpClient.Builder as below:
Sample OkHttpStack class and usage:

```
public class OkHttpStack extends BaseHttpStack {
...
OkHttpClient.Builder clientBuilder = new OkHttpClient.Builder();
clientBuilder.addInterceptor(new AkaOkHttpInterceptor());
OkHttpClient client = clientBuilder.build();
...
}

// Volley queue registration
Volley.newRequestQueue(ctx.getApplicationContext(), new OkHttpStack());
```

5.4 Custom Event Tracking

Custom events are actions triggered due to some activity performed by the end user, such as a button click. The SDK provides APIs that are helpful for developers to clock one or more custom events.

We classify events as timed and instantaneous events. Timed events are ones that have a start and an end point associated with them. Apart from clocking the duration between start and stop, such events can be used to determine network calls originated between start and stop. Please ensure not to add any data to the event name that has privacy implications.

```
/**
* API method to track timed events. Caller must call {@link #stopEvent(String)} to mark an
event as complete.
* @param eventName Name of the event to track. Same name should be used for {@link
#stopEvent(String)}
*/
public void startEvent(String eventName);

/**
* API method to track timed events. Caller must call {@link #startEvent(String)} before
calling this method.
* @param eventName Name of the event used during {@link #startEvent(String)}
*/
public void stopEvent(String eventName);
```

Instantaneous events, unlike timed events, do not have a start and stop associated with them. Such events can be used to log a set of sequences or form a timeline of operations. Note that instantaneous events are in Tech Preview and are not yet displayed in the portal.

```
/**
* API method to track instantaneous events.
* @param eventName Name of the event to track.
*/
public void logEvent(String eventName);
```

Examples

Sample usage of above webview and user events API.

```
import com.akamai.android.sdk.AkaMap;

...
vocService.logEvent("Initialization");
...
vocService.logEvent("Clicked XYZ.com");
String url = "http://www.xyz.com/abc.html";
WebView webView;
...
webView.setWebViewClient(new AkaWebViewL21Client() {
        @Override
        public void onPageStarted(WebView view, String url, Bitmap favicon) {
                super.onPageStarted(view, url, favicon);
                // url loading started.
                vocService.startEvent(url);
        }

        @Override
        public void onPageFinished(WebView view, String url) {
                super.onPageFinished(view, url);
                // url loading finished.
                vocService.stopEvent(url);
        }
});
```

5.5 Network Aware Experience

The SDK provides APIs that help a developer to access client side network quality state to augment client requests.

For example - Load the absolute minimum needed for a particular user event. The example below shows how loading of a web page can be tweaked based on network quality state.

```
import com.akamai.android.sdk.AkaMap;
import com.akamai.android.sdk.MapNetworkQualityStatus;

...

int networkQualityState = AkaMap.getInstance().getNetworkQuality();

switch(networkQualityState) {
	case MapNetworkQualityStatus.POOR:
		// Load a webpage with no images.
		break;
	case MapNetworkQualityStatus.GOOD:
		// Load a webpage with medium quality images.
		break;
	case MapNetworkQualityStatus.EXCELLENT:
		// Download content.
		break;
}
```

5.6 Customer Pinned Certificates

The sdk can be configured to use pinned certificates (SSL Socket Factory) and/or custom hostname verifiers to download prepositioned content. The certificates and hostname verifier can be configured on a per host basis and should be done before registering since the content starts downloading automatically after registration is successful. If configured, the sdk will use the authentication parameters while connecting to the host part in the prepositioned url.

```
import com.akamai.android.sdk.model.MapConnectionParameters


// Create a VocService instance

MapConnectionParameters parameters = new MapConnectionParameters();

paramaters.setSSLSocketFactory(yourCustomSocketFactory);
parameters.setHostnameVerifier(yourCustomHostnameVerifier);

AkaMap.getInstance().setCustomConnectionParameters("www.akamai.com", parameters);
```

5.7 Debugging APIs

SDK provides APIs for developers to debug requests made through SDK. There are two logging levels supported viz. *DEBUG* and *INFO*. *INFO* is the default logging mode and contains logging with minimal output. *DEBUG* on the other hand is the enhanced logging mode. Logging levels can be changed at runtime and are not persisted through multiple app sessions. All the APIs are supported through *Logger.java* class.

```
import com.akamai.android.sdk.Logger;


public enum LEVEL {
  /**
   * The default logging mode. This is the production level with minimal output.
   */
  INFO,
  /**
   * The enhanced logging mode for DEBUGGING purposes only.
   */
  DEBUG
}

/**
*
* @param level defines the SDK logging level.
*                  The default level is LEVEL.INFO. This is the production level with minimal
output.
*                  LEVEL.DEBUG is enhanced logging mode for DEBUGGING purposes only.
* Also see, {@link Logger.LEVEL}
*/
public static void setLevel(LEVEL level)
```

5.8 MAP SDK info

Developers should be using MapSdkInfo class to get the information about the SDK such as is SDK enabled, config info, content info and subscribed segments. These APIs are restricted to be used only when the log level is set to LEVEL.**DEBUG**.

```
import com.akamai.android.sdk.internal.MapSdkInfo;


/**
* @return Subscribed User segments
*/

public static String getSegments()

/**
* @return true if sdk is active
*/
public static boolean isEnabled()

/**
*
* @param ctx
* @throws Exception if current log level is not set to LEVEL.DEBUG.
*/
public static void logCurrentConfiguration(Context ctx) throws Exception {

/**
* Logs content corresponding to all the segments.
* @param ctx
* @throws Exception if current log level is not set to LEVEL.DEBUG.
*/
public static void logExistingContent(Context ctx)
```

5.9 Using SDK's AkaURLStreamHandler

If the application or a library within the app is setting its own global URLStreamHandlerFactory, then sdk provides a solution where developers can use `AkaURLStreamHandler` for specific requests that need to be handled by MAP

```
final String uri = "http://www.bestbuy.com/";

//Before
URL url = new URL(uri);

//After
URL url = new URL(null, uri, new AkaURLStreamHandler());
```

```
// HttpURLConnection usage.

final String uri = "http://www.bestbuy.com/";

HtttpURLConnection urlConnection = null;
try {
        // AkaURLStreamHandler that instantiates an object of URLConnection.
        URL url = new URL(null, uri, new AkaURLStreamHandler());
        urlConnection = (HttpURLConnection) url.openConnection();
        ...
        // Download content using the InputStream
        InputStream inputStream = new BufferedInputStream(urlConnection.getInputStream());
        ...
        // Close the stream once done with the download.
        inputStream.close();
        } catch (IOException e) {
                e.printStackTrace();
        } finally {
 // Make sure to call HttpURLConnection#disconnect() to release resources and collect stats.
                if (urlConnection != null) {
                                urlConnection.disconnect();
                }
        }
```

Here is a side-by-side comparison of API usage for HttpURLConnection

Before	*After*

`final String uri = "http://www.bestbuy.com/";`
 	
`HttpURLConnection urlConnection = null;`
`try {`
`URL url = new URL(uri);`
	`final String uri = "http://www.bestbuy.com/";`
	
	`HttpURLConnection urlConnection = null;`
	`try {`

```
urlConnection = (HttpURLConnection)
url.openConnection();
        ...
InputStream inputStream = new
BufferedInputStream(urlConnection.getInputStr
eam());
        ...
inputStream.close();
} catch (IOException e) {
e.printStackTrace();
} finally {
if (urlConnection != null) {
        urlConnection.disconnect();
}
}
```

```
URL url = new URL(null, uri, new
AkaURLStreamHandler());

urlConnection = (HttpURLConnection)
url.openConnection();
        ...
InputStream inputStream = new
BufferedInputStream(urlConnection.getInputStr
eam());
        ...
inputStream.close();
} catch (IOException e) {
e.printStackTrace();
} finally {
if (urlConnection != null) {
        urlConnection.disconnect();
}
}
```

For all https requests use `AkaURLStreamHandler(true)`

```
final String uri = "https://www.akamai.com/";

HttpsURLConnection urlConnection = null;
try {
        // AkaURLStreamHandler(true) that instantiates an object of AkaSURLConnection.
        URL url = new URL(null, uri, new AkaURLStreamHandler(true));
        urlConnection = (HttpsURLConnection) url.openConnection();
       // set a custom ssl socket factory if needed.
       urlconnection.setSSLSocketFactory(customFactory);

        ...
        // Download content using the InputStream
        InputStream inputStream = new BufferedInputStream(urlConnection.getInputStream());
        ...
        // Close the stream once done with the download.
        inputStream.close();
        } catch (IOException e) {
                e.printStackTrace();
        } finally {
 // Make sure to call HttpsURLConnection#disconnect() to release resources and collect
stats.
```

```
            if (urlConnection != null) {
                        urlConnection.disconnect();
            }
      }
```

5.10 Cache-Control request parameters

MAP SDK ensures delivery of fresh content. Content is either served from the cache or from the network transparently. In certain cases, it may be desirable to override this behavior. For ex - In case of poor connectivity, a caller may be okay to use stale responses for a particular request(s). Or in certain cases, a caller may want the content that's being served (in case of cached content) to be revalidated by origin server by controlling its expiry or forcing a revalidation in a particular scenario such as a certain time of day.
MAP SDK provides following API for this:

```
/**
*
* @param field non-null key
* @param value non-null value<br>
* SDK cache behavior can be controlled with use of following key-value pairs. <br>
*
*          Pragma:no-cache: Forces SDK to revalidate cached response.<br>
*          Cache-Control:no-cache: Same as Pragma:no-cache.<br>
*          Cache-Control:max-age='x': Forces SDK to select expiry for the content as Min('x', expiry calculated from response headers)<br>
*          Cache-Control:max-stale='x': If assigned a value, then 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.<br><br>
*
*          Note: max-age/max-stale is ignored if no-cache is present.
*
*
*/
@Override
public void setRequestProperty(String field, String value);
```

6 QUIC Library Integration

MAP sdk has a capability to accelerate requests using chromium's QUIC protocol. If the configuration to enable quic from the portal is turned on, all the requests by default will try to use QUIC and if the server supports QUIC the response is served over QUIC else it will fall back to HTTP. To enable this capability on the app, an additional aar file needs to be included as below

- Download the SDK and unzip it.
- In a file explorer (*not* Android Studio), drag the unzipped akamai-cronet-lib-release-version.aar file into the /app/libs directory in your project's root directory.
- In Android Studio, edit the build.gradle file in the **app** directory (*not the one in the root folder*) and edit the dependencies sub-section to include .AAR file and following library

```
dependencies {
        implementation(name:'akamai-cronet-lib-release-<version>', ext:'aar')
        ...
}
```

If you build infrastructure of your application is not using Java 8, the code below needs to be added to the build.gradle file

```
android {
     compileOptions {
   sourceCompatibility 1.8
   targetCompatibility 1.8
}
}
```

ABI Management for different architectures

Akamai cronet aar file is around 4.8MB and includes supports two different architectures

- armeabi-v7a
- arm64-v8a

As the size of SDK is always a priority, these architectures can be selectively filtered based on the target sdk version and audience of the host application using abiFilters in the build.gradle file. For instance, just including armeabi-v7a is enough to support the majority of the devices,except v64 devices. So based on the audience of the host application these architectures can be selectively chosen.

Example : just including "armeabi-v7a" architecture (Recommended for most cases)

```
android{
defaultConfig{ ndk{ abiFilters "armeabi-v7a" } }
}
```

7 Brotli Library Integration

MAP sdk has a capability to accelerate requests using Brotli compression. Brotli provides a much denser compression of the data when compared to gzip. If the configuration to enable brotli from the portal is turned on, all the requests by default will try to use brotli encoding by including it in the 'accept-encoding' header. If the server supports brotli and responds with a brotli encoded stream, it will be decoded by the sdk and provided to the application. In case the requirements are not met, default 'gzip' encoding is used. To enable this capability on the app, an additional aar file needs to be included as below

- Download the SDK and unzip it.
- In a file explorer (*not* Android Studio), drag the unzipped akamai-brotli-lib-release-<version>.aar file into the /app/libs directory in your project's root directory.
- In Android Studio, edit the build.gradle file in the **app** directory (*not the one in the root folder*) and edit the dependencies sub-section to include .AAR file and following library

```
dependencies {
        implementation(name:'akamai-brotli-lib-release-version', ext:'aar')
        ...
}
```

8 mPulse Integration

When integrating MAP and mPulse, ensure the client uses the same version for both map and mPulse dependencies. aka-common is a library used by both SDKs and gradle will pick the latest aka-common automatically.

```
//Assuming you are using gradle
dependencies {
        implementation 'com.akamai.android:aka-map:21.60.+'
        implementation 'com.akamai.android:aka-mpulse:21.60.+'
        ...
}
```

When using both MAP and mPulse, the clients that are using okhttp wrappers should be using AkaOkHttpInterceptor (or AkaOkHttpAppInterceptor) under the MAP/wrappers directory.

9 Managing Cookies

With a multitude of third party libraries available for network request and image downloading (such as OkHttp, Picasso, Glide) as well as the combined use of android WebView and HttpUrlConnection, managing cookies becomes an essential part of any app. Please note that if the java.net.CookieManager is already set up in the app, there is nothing else to do and this section may be skipped. This part serves as a general guidance for managing the cookies for MAP-SDK use. The code snippets here are for example only.

- The MAP SDK uses android HttpUrlConnection to send the request. The HttpUrlConnection uses java.net.CookieManager to get and save the cookies for a session. Once the cookie manager is created, the cookies are automatically saved from the response and appended to the subsequent requests (applying all the rules for appending the cookie). Here is what the app can do to create a new cookie manager.

```
CookieManager cookieManager = new CookieManager();
cookieManager.setCookiePolicy(CookiePolicy.ACCEPT_ALL);
CookieHandler.setDefault(cookieManager);
```

This cookieManager will by default use an in memory cookie store. Optionally, the cookie manager can be created with a custom implementation of the Cookie store that would persist the cookies across app restarts, for example.

```
CookieManager cookieManager = new CookieManager(new CustomCookieStore(),
CookiePolicy.ACCEPT_ALL);
```

- The OkHttp client uses a CookieJar. In case the app wants to continue to use the CookieJar, it can be created in the following way to make the cookies also available to the HttpUrlConnection.

```
CookieManager cookieManager = new CookieManager();
cookieManager.setCookiePolicy(CookiePolicy.ACCEPT_ALL);
CookieHandler.setDefault(cookieManager);
CookieJar cookieJar = new JavaNetCookieJar(cookieManager);

dependencies {
       implementation "com.squareup.okhttp3:okhttp-urlconnection:3.12.0"
```

```
        ...
}
```

Any cookies now added to the cookie jar will also be accessible to the MAP-SDK.

- Syncing cookies between the web views and the httpUrlConnection requests: The cookies for HttpUrlConnection are maintained by java.net.CookieManager while the cookies for Webview are maintained by android.webkit.CookieManager. If the app is using a combination of Webview and Net (HttpUrlConnection) requests, the app needs to manually synchronize the cookies between the two managers.

  Here is a code example to synchronize the cookies from java.net.CookieManager to android.webkit.CookieManager for a request to 'uri' after getting a response.

```
CookieHandler handler = CookieHandler.getDefault();
Map<String, List<String>> cookie = handler.get(uri, responseHeaders);
android.webkit.CookieManager.getInstance().setCookie(uri, cookie.get("Cookie").get(i));
```

- Picasso, Volley and Glide: These packages don't have any specific cookie handling. Once the cookie Manager is created and set as in section 1, the intercepted requests by MAP-SDK will save and append the cookies.

9 Appendix - Requirements and Dependencies

9.1 Background Execution

The MAP SDK downloads any prepositioned 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. MAP SDK also downloads content(marked for prepositioned) in background triggered by FCM push notifications. There is no additional certificate configuration needed for push notifications to work

9.2 Upgrading from a previous SDK version to 20.3.2 and above

Please follow the 20.3.2 upgrade guide packaged with the zip file.
MAP SDK - Android v20.32+ Upgrade Guide.pdf

9.3 SDK Debug logs

For us to debug an SDK issue, we need logs. Retrieve us the logs associated with the following tag

```
adb logcat -s 'AkaSdKLogger'
```

If you are using both MAP and mPulse, use the following to verify the logs

MAP

```
adb logcat -s 'AkaSdKLogger-map'
```

mPulse

```
adb logcat -s 'AkaSdKLogger-mpulse'
```

common

```
adb logcat -s 'AkaSdKLogger-common'
```

9.4 Proguard

Proguard rules are already applied for MAP and customers do not have to apply one. The one marked with *@PublicApi* are the classes and methods that are available for customer use. The one marked with *@AkamaiInternal* is not for customer use. If you make any calls to those internal methods, you will receive a lint error.

9.5 WorkManager Dependency

MAP SDK has work manager dependency and it is automatically pulled into your project. Gradle will resolve and pick up the latest version of Work Manager for your project. If you prefer not to use the one

provided by MAP SDK and use a different one in your project, please exclude the work manager from MAP SDK.

```
dependencies {
       implementation ('com.akamai.android:aka-map:21.60.+') {
          exclude group: "androidx.work", module: "work-runtime"
       }
}
```

9.5 Troubleshooting guide

You need to set the log level to debug to see the debug logs associated with map-sdk

```
Logger.setLevel(Logger.LEVEL.DEBUG);
```

Success

SDK discovered

```
D/AkaSDKLogger: Initialized com.akamai.android.aka_common, version
D/AkaSDKLogger: ComponentContainer: Initialized com.akamai.android.sdk,
D/AkaSDKLogger: AkaInitProvider: AkaCommon initialization successful through
CP.
```

Sync started

```
D/AkaSDKLogger: AkaBackgroundService: com.akamai.anaina.FULL_SYNC
```

Interception started:

```
D/AkaSDKLogger: External Url Stream handler to handle http/s stream::
com.akamai.android.sdk.AkaMap
```

Config requested

```
D/AkaSDKLogger: Request headers for Url:
https://configuration-map.akamai.com/config?id=
```

Config received

```
D/AkaSDKLogger: d: Capabilities: {"license_hash":
```

Prepositioning triggered

```
D/AkaSDKLogger: AnaWebContentDownloader: WebContent: Queued for download  11,
policy  0
```

Analytics success

```
D/AkaSDKLogger: AkaWebAnalyticsHandler: sendWebAccAnalytics: 200
```

Push notification

```
D/AkaSDKLogger: PushMessagingService: Fcm rcv Prepare sync
```

Failures

Wrong library

```
ERROR: Failed to resolve: com.akamai.android:aka-map:20.32.120
```

Solution: Integration step is wrong. Add jcenter() in the project build.gradle

Missing config file reference

```
E/AkaSDKLogger: Couldn't find resource file for meta-data key
com.akamai.android.sdk!
```

Solution: Check *akamai_sdk_init.xml* file is in **res/xml** and reference it in the AndroidManifest.xml

```
<meta-data
            android:name="com.akamai.android.sdk"
            android:resource="@xml/akamai_sdk_init" />
```

Wrong key

```
E/AkaSDKLogger: AkaConfigHandler: getConfig with
exception:java.io.FileNotFoundException:
E/AkaSDKLogger: AkaSyncController: sync: SDK is not active
```

Solution: Ensure that licenseKey is added in *akamai_sdk_init.xml*.

```
<com_akamai_sdk_license_key>eb01fd0959b8b4xxxxxxxxxxxxxxxx9f91c978770daa44ea903fc4808c20e5cf</co
m_akamai_sdk_license_key>
```

Verifying Content Logs

- **Content served from Universal Cache**
    - D/AkaSDKLogger: D/AkaSDKLogger: AkaURLConnection: Stats: URL: https://www.akamai.com/, Type: CACHE_FETCH_ADHOC, Connection: cellular/LTE, RespCode: 200, ContentLength: 251167, StartTime: 1523471307223, Duration: 55, Ttfb: 6

- **Content served from Preposition Cache**

- D/AkaSDKLogger: D/AkaSDKLogger: AkaURLConnection: Stats: URL: https://www.akamai.com/, Type: CACHE_FETCH, Connection: cellular/LTE, RespCode: 200, ContentLength: 251167, StartTime: 1523471307223, Duration: 55, Ttfb: 6

- **Content served from Network**
  - D/AkaSDKLogger: D/AkaSDKLogger: AkaURLConnection: Stats: URL: https://www.akamai.com/, Type: CACHE_MISS, Connection: cellular/LTE, RespCode: 200, ContentLength: 251167, StartTime: 1523471307223, Duration: 55, Ttfb: 6

FullBackupContent cause manifest merge to fail

```
Manifest merger failed : Attribute application@fullBackupContent
value=(@xml/webengage_backup_rules) from AndroidManifest.xml:28:9-64
is also present at [com.akamai.android:aka-map:21.12.14] AndroidManifest.xml:16:18-67
value=(@xml/map_backup_rules).
```

Solution: If you have your own backup rules specified, merge them with MAP SDK rules manually by adding the following rule:

```
<exclude domain="sharedpref" path="com.akamai.android.sdk.PCDPreferences.xml"/>
```

Add **tools:replace="android:fullBackupContent"** to the **Application** element in the **Manifest** file so that the manifest merger picks app's backup rules instead:

```
<application
android:allowBackup="true"
android:fullBackupContent="@xml/my_backup_rules"
tools:replace="android:fullBackupContent"
```

10 Release Notes

The MAP Android SDK can be found on JCenter.

21.60 (2020-09-17)

Changes:

- Bug fixes
- Update Work Manager to 2.4.0 version

21.50 (2020-08-13)

Changes:

- Bug fixes

21.40 (2020-06-30)

Changes:

- Bug fixes

21.30 (2020-05-29)

Changes:

- Bug fixes
- Upgrade cooperation with 3rd party SDK’s

21.21 (2020-04-27)

Changes:

- Bug fixes

21.13 (2020-04-07)

Changes:

- Bug fixes

21.12 (2020-02-14)

Changes:

- Supporting alternate sources per segment.
- Supporting network selection per segment.
- Bug fixes

21.11 (2020-01-21)

Changes:

- Segments will be downloaded based on priority.
- Bug fixes

20.41 (2020-01-09)

Changes:

- Migrated from Intent Service to Work Manager
- Bug fixes

20.33 (2019-11-20)

Changes:

- Migrated to androidx libraries
- Moved to proguard from R8
- Bug Fixes on whitelist Filters.
- Added wrapper to support Picasso 2.7+

20.32 (2019-10-14)

- New control path has been introduced which helps map-sdk to be very agile and be cache ready in a few seconds.
- New architecture has been introduced which allows customers to use the combination of map-sdk and mpulse-sdk.
- Simplified the integration approach.
- Several Bug Fixes

DEPRECATIONS

Major release where most of the existing APIs are deprecated and new ones are introduced. Important deprecations are

- VocService
- VocConfigBuilder