Marmalade Extension v1.0

SDK Integration Whitepaper

Ver 1.0 | January 9, 2012

Prepared by the **MobileAppTracking Engineering Team**

Introduction & Table of Contents

Marmalade Extension v1.0

The MobileAppTracking (MAT) extension for Marmalade provides basic application install and event tracking functionality. To track installs, you must integrate the Marmalade edk extension with your Marmalade app. Once the SDK is integrated and set to track installs, you can add and track additional events beyond an app install (such as purchases, game levels, and any other user engagement).

This document outlines the Marmalade SDK integration and use cases.

CONTENTS

1. Getting Started
2. Build the Marmalade MAT Extension
3. Test Application
4. Sample Project Code
5. Sample Workflow

1. Getting Started

To use the Marmalade extension for the MAT SDK, you use the extension built in the extension build section. The sample project tests various methods of the MAT SDK either via the Android SDK or the iOS SDK. The MAT Marmalade extension is build on top of the latest MAT SDKs (jar file for Android, static .a or library file for iOS).

2. Build the Marmalade MAT Extension

This section assumes the following base folder: ios_android

Files

s3eMATSDK.s4e
- This is the definition file for all of the extension information such as structures, callbacks and methods

s3eMATSDK.mkf
- Make file to create the extension, defines what files to use and linker options for the extension. This is also where the MATSDK framework files are defined that need to be linked in.

s3eMATSDK_iphone/android.mkb
- Describes the resulting library that combines MATSDK static library with Marmalade methods to create a new static library

s3eMATSDK_build.mkf
- describes where the .o files from the MATSDK static library come from to combine to make the s3eMATSDK.a file, this will be the place where the android source gets linked in as well

s3eMATSDK_android_java.mkb
- Auto-generated java make file

To Build the extension for iOS

The MobileAppTracking iOS static library must have it's object files extracted and then combined and build into the .s4e Marmalade extension file.

1. create or add/modify prototype methods in the .s4e file
2. extract the .o files from the MATSDK .a file, these files will be used when building the edk library
   create s3eMATSDK/incoming folder, this folder will be used as input when building the extension file
   extract a slice from the MATSDK framework fat file (MobileAppTracker.a) into a new .a file from the /incoming folder
   lipo MobileAppTracker.a -thin armv7 -output MobileAppTrackerArmv7.a
   extract the .o files from the MobileAppTrackerArmv7.a file
   ar -x MobileAppTrackerArmv7.a
3. rebuild the extension files:
   from ios_android folder, run:
   /Developer/Marmalade/6.2/s3e/lib/python/run_python /Developer/Marmalade/6.2/s3e/edk/builder/edk_build.py s3eMATSDK/s3eMATSDK.s4e --platform=iphone
   - creates:

- h/s3eMATSDK.h
- interface/s3eMATSDK_interface.cpp
- source/generic/s3eMATSDK_register.cpp

4. The following files should be edited manually since step 2 won't change them:
   - source/generic/s3eMATSDK.cpp <<--- add new methods from the .s4e file header
   - source/h/s3eMATSDK_internal.h <<--- add new methods from the .s4e file header
   - source/iphone/s3eMATSDK_platform.mm <<--- this is the file that calls the native mobileapptracker static library
5. Build the library files to be used in the test app
   /Developer/Marmalade/6.2/s3e/bin/mkb s3eMATSDK/s3eMatSDK_iphone.mkb --arm

To Build the extension for Android

1. create or add/modify prototypes in the .s4e file
2. rebuild the extension files:
   /Developer/Marmalade/6.2/s3e/lib/python/run_python /Developer/Marmalade/6.2/s3e/edk/builder/edk_build.py s3eMATSDK/s3eMATSDK.s4e --platform=android

- creates:
  - h/s3eMATSDK.h
  - interface/s3eMATSDK_interface.cpp
  - source/generic/s3eMATSDK_register.cpp

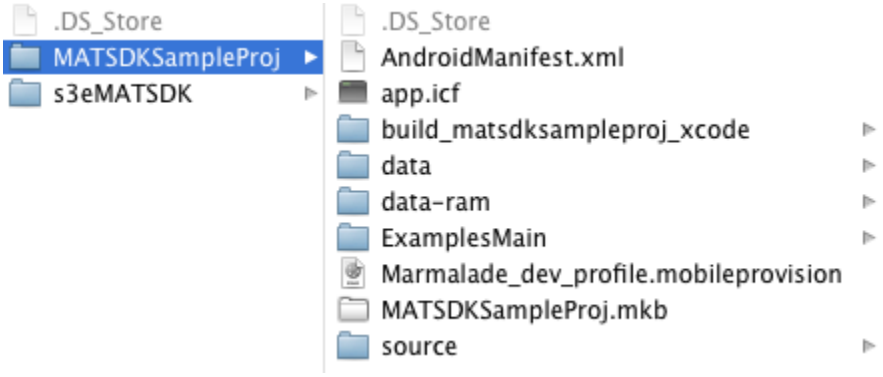
3. The following files should be edited manually since step 2 won't change them:
   - source/generic/s3eMATSDK.cpp
   - source/h/s3eMATSDK_internal.h
   - source/android/s3eMATSDK_platform.cpp <<--- add new or changed methods here, these pass thru to the .java file
   - source/android/s3eMATSDK.java <<-- this is the code that actually calls the mobileapptracker.jar file
4. Build the library files to be used in the test app
   set the NDK environment variable in terminal:
   export NDK_ROOT="/Developer/android-ndk-r8c"
   run:
   /Developer/Marmalade/6.2/s3e/bin/mkb s3eMATSDK/s3eMatSDK_android.mkb --arm
5. Build the java .jar file
   /Developer/Marmalade/6.2/s3e/bin/mkb s3eMATSDK/s3eMatSDK_android_java.mkb

3. Test Application

In /ios_android/MATSDKSampleProj <<--- same project for both android and ios

The project uses Marmalade's ExamplesMain to draw on the screen. The test application will exercise all of the major methods of the MobileAppTracker SDK. It's set up to run on a demo account, but can be modified to point to any account.

Test application project folder:



iOS

1 Testing with a static library for iOS cannot be run on the simulator, it won't load the MATSDK library
2 The MATSDKSampleProj.cpp calls the _platform extension methods and combines the MAT sdk delegate callback.
3 Run the MATSDKSampleProj.mkb (double click) to create and open the sample project in xcode
4 The xcode project provides a unit test to exercise the MAT SDK edk methods

Android

1. Testing can occur on both device or emulator
2. in /ios_android/s3eMATSDK/source/android
3. s3eMATSDK.java - contains the code to directly call the mobileapptracker jar code.

Build and run the iPhone test app

1. Requires a device to run on
2. Build the .ipa file in MATSDKSampleProj folder:

/Developer/Marmalade/6.2/s3e/bin/mkb MATSDKSampleProj.mkb --arm --debug --deploy=iphone --verbose=2

3. Double click on the .ipa file to install via iTunes

ios_android/MATSDKSampleProj/build_matsdksampleproj_xcode/deployments/default/iphone/debug/MATSDKSampleProj.ipa

4. If the .ipa file won't build, it has to be signed with a valid developer provisioning profile

Build and run the Android test app

1. can run on simulator or device
2. build the .apk from MATSDKSampleProj:

/Developer/Marmalade/6.2/s3e/bin/mkb MATSDKSampleProj.mkb --arm --debug --deploy=android --verbose=2

3. in

/ios_android/MATSDKSampleProj/build_matsdksampleproj_xcode/deployments/default/android/debug/arm

How to install the .apk on the device or simulator (note, replace path with your android tools path)

~/Documents/android-sdk-macosx/platform-tools/adb kill-server ~/Documents/android-sdk-macosx/platform-tools/adb start-server ~/Documents/android-sdk-macosx/platform-tools/adb get-state ~/Documents/android-sdk-macosx/platform-tools/adb install -r MATSDKSampleProj.apk

4. Sample Project Code

Here are a few code samples from the Marmalade extension and the Sample Project that tests the extension.

.s4e File

The Extension begins with a .s4e file that describes the methods and data structures that will be implemented by the extension. The methods are platform independent as by this example:

```
void s3etrackInstallWithReferenceId(const char* refId) run_on_os_thread
```

The .s4e file is used by Marmalade to create the source files that will be used in creating platform code.

MobileAppTracker SDK methods implemented by Marmalade Extension. Please see the MobileAppTracker.h file for further information. Also, note, not all of these methods are implemented in all platforms.

```
void s3eStartMobileAppTracker(const char* adId, const char* adKey);
void s3eSDKParameters();
void s3etrackInstall();
void s3etrackUpdate();
void s3etrackInstallWithReferenceId(const char* refId);
void s3etrackActionForEventIdOrName(const char* eventIdOrName, bool isId, const char* refId);
void s3etrackActionForEventIdOrNameItems(const char* eventIdOrName, bool isId, const
s3eMATArray* items, const char* refId, double revenueAmount, const char* currencyCode, uint8
transactionState);
void s3eSetPackageName(const char* packageName);
void s3eSetCurrencyCode(const char* currencyCode);
void s3eSetDeviceId(const char* deviceId);
void s3eSetOpenUDID(const char* openUDID);
void s3eSetUserId(const char* userId);
void s3eSetRevenue(double revenue);
void s3eSetSiteId(const char* siteId);
void s3eSetTRUSTeId(const char* tpid);
void s3eSetDebugResponse(bool shouldDebug);
```

5. Sample Workflow

Starting with the sample project, in the MATSDKSampleProj.cpp, there are a series of buttons that execute various methods. For this example, we'll follow the flow of the track install method (trackInstallWithReferenceId).

1 Button press for Track Install button:
calls s3etrackInstallWithReferenceId("Marmalade Install Test");
this calls a method in the s3eMATSDK->Source->Generic->se3MATSDK.cpp. The generic cpp file then passes the method to the appropriate platform file.

```
void s3etrackInstallWithReferenceId(const char* refId)
{
   s3etrackInstallWithReferenceId_platform(refId);
}
```

this calls a method in either the android or iphone platform code:

for iphone:
in se3MATSDK_platform.mm, this actually calls the MobileAppTracker static library in iOS:

```
void s3etrackInstallWithReferenceId_platform(const char* refId)
{
        NSLog(@"track install %@", [NSString stringWithUTF8String:refId]);
        [[MobileAppTracker sharedManager] trackInstallWithReferenceId:[NSString
stringWithUTF8String:refId]];
}
```

for android:
the platform file s3eMATSDK_platform.cpp uses JNI to call java methods described in the s3eMATSDK.java file:
In the init method se3Result s3eMATSDKInit_platform(), using JNI, the install method is put into a method reference variable:

```
g_s3etrackInstallWithReferenceId = env->GetMethodID(cls,
"s3etrackInstallWithReferenceId", "(Ljava/lang/String;)V");
```

Then, the method variable is called via this code:

```
void s3etrackInstallWithReferenceId_platform(const char* refId)
{
        JNIEnv* env = s3eEdkJNIGetEnv();
        jstring refId_jstr = env->NewStringUTF(refId);
        env->CallVoidMethod(g_Obj, g_s3etrackInstallWithReferenceId, refId_jstr);
        env->DeleteLocalRef(refId_jstr);
}
```

This is then passed through to the actual java code in s3eMATSDK.java:

```
public void s3etrackInstallWithReferenceId(String refId)
{
mat.setRefId(refId);
mat.trackInstall();
}
```

This is where the actual call to the MobileAppTracker.jar file occurs.

Testing the iOS or Android app shows the following:

Calling the Install Method: