MobileFirst Quality Assurance for native iOS applications

Overview

Prerequisite: This tutorial is a continuation of the MobileFirst Quality Assurance (../overview/) tutorial. To follow it, you must first set up IBM Mobile Quality Assurance (MQA).

The tutorial explains how to install and configure the Mobile Quality Assurance client SDK for iOS, and how to enable apps to send back bug reports and feedback.

Jump to:

Creating a simple iOS app

If you do not have an app that you would like to use for this procedure, you can create a simple app by completing the following steps:

1. Create a new project by using Xcode Project.
2. Select a **Single view application**.
3. Name your new application and select where you want to store your project files.
4. Select your preferred programming language, and select iPhone or universal for the device type.
5. Build and test your app by running it in the simulator.

Installing the iOS libraries

To run MQA with your iOS app, you must download and install the MQA iOS SDK, as follows:

1. Prepare the iOS SDK.
   Important: If you are using Xcode 7.x on iOS 9 and are using the Mobile Quality Assurance iOS SDK version 2.4.1 or earlier, you must set the Enable Bitcode setting to *No* in the Xcode Build Settings.
   1. Create a `frameworks` folder.
   2. Download the most recent version of the Mobile Quality Assurance SDK from Device SDKs for Mobile Quality Assurance for Bluemix (http://www.ibm.com/support/docview.wss?uid=swg27044490).
      For more information about downloading the SDKs, see Installing the iOS libraries in Objective-C (http://www.ibm.com/support/knowledgecenter/SSJML5_6.0.0/com.ibm.mqa.uau.saas.doc/topics/t_InstallingTheIOSLibraries.html) or Installing the iOS libraries in a Swift app (http://www.ibm.com/support/knowledgecenter/SSJML5_6.0.0/com.ibm.mqa.uau.saas.doc/topics/t_Installing_IOS_Swift.html).

      Important: Ensure that you download the SDK that corresponds to your programming language.

   3. Extract the contents of the SDK, which includes the framework named `Q4M.framework`.
2. Add the iOS SDK to your project.
   1. Drag the `Q4M.framework` folder into the Frameworks group in the navigation tree.
   2. In the Choose options window, select **Create groups**, and select your iOS app as the target.
   3. In the Xcode Project Navigator, click your project and select **Build Phases**.
   4. Click **Link Binary With Libraries**.
   5. Add the following frameworks by clicking the plus (+) symbol in the "Link Binary With Library" section:
      - `CoreTelephony.framework`
      - `AssetsLibrary.framework`
      - `Security.framework`
      - `SystemConfiguration.framework`
      - `AudioToolbox.framework`
      - `CoreData.framework`
      - `CoreLocation.framework`
      - `CoreMedia.framework`

- `CoreTelephony.framework`
- `CoreText.framework`
- `CoreVideo.framework`
- `CorePlayer.framework`
- QuartzCore.framework
- Security.framework
- SystemConfiguration.framework
- `CoreMotion.framework`

6. Set the linker flags for Debug and Release to `-ObjC`.
    1. Click your project in the Project Navigator.
    2. Click **Build Settings**.
    3. Enter `Other Linker Flags` in the Search field.
    4. In the **Other Linker Flags** section, expand the rows for Debug and Release.
    5. Enter the following line in both the Debug and Release rows: `-all_load -ObjC`
7. For Swift apps only: Add an Objective-C bridging header file.
    1. Right-click the project root node and select **New file**.
    2. In the Choose Template window with iOS section, click **Source**, and then click the template for a Header File.
    3. Name the file *`project_name`*`-Bridging-Header.h`, where *project_name* is the name of your Swift project file.
    4. Select your project as the target.
    5. Click **Save** to save the file in the root of your project.
    6. Add the following line to the bridge header file that you created: `#import`
    7. Click **Build Settings** in the Xcode Project Navigator, and search for **Objective-C Bridging Header**.
    8. In the Swift Compiler-Code Generation section, set the value of Objective-C Bridging Header to the path of your bridging header file: `$(SRCROOT)/$(PROJECT_NAME)/-Bridging-Header.h`
    9. Save and build the app to verify that the path is correct.

Configuring how MQA communicates with your app

1. Set the mode to either production or preproduction mode.
    - Swift:
        1. In the `AppDelegate.swift` file, locate the `didFinishLaunchingWithOptions` method.
        2. To use preproduction mode, add the following line to the `didFinishLaunchingWithOptions` method: `MQALogger.settings().mode = MQAMode.QA`
        3. To use production mode, add the following line to the `didFinishLaunchingWithOptions` method: `MQALogger.settings().mode = MQAMode.Market`
    - Objective-C:
        1. Add the following import statement to the `AppDelegate.m` and `ViewController.m` files: `#import`
        2. In the `AppDelegate.m` file, locate the `didFinishLaunchingWithOptions` method.
        3. To use preproduction mode, add the following line to the `didFinishLaunchingWithOptions` method: `[[MQALogger settings] setMode:MQAModeQA];`
        4. To use production mode, add the following line to the `didFinishLaunchingWithOptions` method: `[[MQALogger settings] setMode:MQAModeMarket];`
2. Launch a new MQA session each time your app starts.
    1. Open the `AppDelegate` file, if it is not already open.
    2. Locate the `didFinishLaunchingWithOptions` method.
    3. Add the following code to the `didFinishLaunchingWithOptions` method to start a new session:
        - Swift: `MQALogger.startNewSessionWithApplicationKey("your-application_key")`
        - Objective-C:[MQALogger startNewSessionWithApplicationKey:@"your-application_key"];

       The *your_application_key* argument is the application key that is provided by your MQA instance.

Testing your app

After the environment is set up, you can load your app on your iOS device and start testing it. Complete the following steps to report a bug or provide feedback during testing:

1. Shake your mobile device lightly.
2. Select **Report a bug** or **Give feedback**.
3. Enter the required information and select **Send** to send the information.
4. View the bug reports and feedback in your Bluemix MQA instance where you retrieved the application key.

For more information about reporting bugs, see Reporting bugs on iOS devices (http://www.ibm.com/support/knowledgecenter/SSJML5_6.0.0/com.ibm.mqa.uau.saas.doc/topics/tReportingBugsOnIOSDevices.html). For more information about submitting feedback, see Enabling feedback on an iOS device (http://www.ibm.com/support/knowledgecenter/SSJML5_6.0.0/com.ibm.mqa.uau.saas.doc/topics/tiosFeedback.html).

Further discovery

After you are comfortable with the basics of setting up MQA with your app, you can read the content in IBM Knowledge Center to learn and explore some of the other things that you can do. For example:

- Managing users and testers (http://www.ibm.com/support/knowledgecenter/SSJML5_6.0.0/com.ibm.mqa.uau.saas.doc/topics/t_ManagingTesters.html). You can manage the roles and access of your team as they apply to your app and MQA instance.
- Distributing and managing builds (http://www.ibm.com/support/knowledgecenter/SSJML5_6.0.0/com.ibm.mqa.uau.saas.doc/topics/t_DistributingBuilds.html). You can distribute test builds and production builds to the appropriate audience.
- Managing reports (http://www.ibm.com/support/knowledgecenter/SSJML5_6.0.0/com.ibm.mqa.uau.saas.doc/topics/t_ManagingReports.html). You can automatically distribute test builds and production builds to your audience.
- User sentiment analytics (http://www.ibm.com/support/knowledgecenter/SSJML5_6.0.0/com.ibm.mqa.uau.saas.doc/topics/UserSentiment.html). You can use MQA to analyze and monitor user comments and information about your app. You can also compare the analysis of those comments with the analysis of the user feedback from similar apps.