ASAPP Framework Integration:

Add ASAPP.framework to Embedded Binaries

Add ASAPP.framework to the desired location in your project's directory. After opening Xcode, drag-and-drop ASAPP.framework into "Embedded Binaries" in the "General" settings of your project's target. Xcode should now display ASAPP.framework under "Embedded Binaries" and "Linked Frameworks and Libraries".

Always Embed Swift Standard Libraries -> Yes

Under your target's "Build Settings", set "Always Embed Swift Standard Libraries" to "Yes". This is necessary because the ASAPP iOS SDK is built using Swift.

User-Defined Setting "SWIFT_VERSION" -> "3.0"

This should not be required for most projects, but if you have used an older version of Swift in your project, you may need to add a user-defined setting to avoid any compiler issues. Go to the "Build Settings" of your target, scroll to "User-Defined" near the bottom, and add a new key "SWIFT_VERSION" with value "3.0".

Add NSPhotoLibraryUsageDescription/NSCameraUsageDescription to Info.plist

As of iOS 10, Apple requires a description for why your app uses the photo library and/or camera. Since our SDK has features using the photo library and the camera, your Info.plist will need brief descriptions ("In-app chat allows users to upload images") for both of these. If accessing the Info.plist via Xcode, the keys are "Privacy - Camera Usage Description" and "Privacy - Photo Library Usage Description". If accessing your Info.plist via a text editor, the keys are "NSPhotoLibraryUsageDescription" and "NSCameraUsageDescription".

Add Custom Run Script to Build Phases

ASAPP.framework is distributed as a fat framework, meaning that it is built to run on all devices and simulators. In order to upload your app to the App Store, you will need to strip the ASAPP.framework of the information used to build it for the iOS Simulator. If you are incorporating other custom frameworks, you may already have a run script phase that handles this. However, if not, we have included a simple run script that will take care of this for you.

Under the "Build Phases" tab of your target, click the plus button to add a "New Run Script Phase". Make sure the "Shell" value is "/bin/sh". Copy-paste the contents of the asapp_run_script.sh file into the script area.

ASAPP SDK Usage

Create an instance of ASAPPButton

ASAPP's chat (SRS) functionality can be used in one of two ways. The first, and simpler of the two, is to create an instance of ASAPPButton and add this to your view. The button will size itself and handle presenting the chat view controller automatically, so all the caller is responsible for is adding the button to a view and setting its position. Alternatively, the caller may use a custom button and handle the view controller presentation by creating an instance of ASAPPChatViewController. With either method, the caller should use the creation methods provided by the ASAPP class.

Since ASAPPButton is a subclass of UIView, you can change the visibility of the button by updating the "hidden" (isHidden) property. This replaces the "viewDidAppear" and "viewWillDisappear" methods from v1.

ASAPPButton

company (string):
This is a unique string that identifies your company. If you are unsure what this is, please contact ASAPP.

customerId (string):
This should be a unique identifier for your customer. This value will be used to pull up the history of a user's conversation.

environment (enum):
This is an enum for determining which environment to use. You can use "ASAPPEnvironmentStaging" for testing on the staging environment, but you should set this value to "ASAPPEnvironmentProduction" before submitting to the App Store.

authProvider (block):
This is called every time the chat is opened and is used to authenticate requests to the server. You will need to pass in an "access_token" (String), and optionally, an "issued_time" (NSDate) and "expires_in" (NSTimeInterval - seconds).

contextProvider (block):
This is called every time the chat is opened. You can pass any context that might be relevant to the conversation here.

callbackHandler (block):
The callbackHandler is called anytime the user taps button that links back to your app.

styles (ASAPPStyles, nullable):
If you'd like to customize the colors/fonts used in the chat view controller, you can create an instance of ASAPPStyles and set the desired values for fonts/colors. This is not needed to create an instance of the ASAPPButton or ASAPPChatViewController.

presentingViewController (UIViewController)
The presentingViewController will be used for modally presenting the chat view controller.

Example ASAPPButton (Objective-C):

```
@import ASAPP;

@interface YourViewController ()
@property (nonatomic, strong) ASAPP *chatButton;
@end

@implementation YourViewController

- (void)viewDidLoad {
      [super viewDidLoad];

      self.chatButton = [ASAPP
                         createChatButtonWithCompany:@“company_name”
                         customerId:@“unique_customer_id”
                         environment:ASAPPEnvironmentStaging
                         authProvider:^NSDictionary * {
                             return @{
                                 [ASAPP AUTH_KEY_ACCESS_TOKEN] : @“sample_token”,
                                 [ASAPP AUTH_KEY_ISSUED_TIME] : [NSDate new],
                                 [ASAPP AUTH_KEY_EXPIRES_IN] : @(3600) // 1 hour
                             };
                         }
                         contextProvider:^NSDictionary * {
                             return @{
                                 @“context_key” : @“context_value”
                             };
                         }
                         callbackHandler:^(NSString *deepLink, NSDictionary *data) {
                             // Handle deepLink + data here
                         }
                         styles:nil
                         presentingViewController:self];

      // Add the button to your view
      [self.view addSubview:self.chatButton];
}

- (void)viewWillLayoutSubviews {
      [super viewWillLayoutSubviews];

      // Position the button
      self.chatButton.center = CGPointMake(100, 100);
}

@end
```

Run-Script for Stripping Unused Frameworks:

```
APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"

# This script loops through the frameworks embedded in the application and
# removes unused architectures.
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist"
CFBundleExecutable)
FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"

EXTRACTED_ARCHS=()

for ARCH in $ARCHS
do
echo "Extracting $ARCH from $FRAMEWORK_EXECUTABLE_NAME"
lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o "$FRAMEWORK_EXECUTABLE_PATH-
$ARCH"
EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
done

echo "Merging extracted architectures: ${ARCHS}"
lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
rm "${EXTRACTED_ARCHS[@]}"

echo "Replacing original executable with thinned version"
rm "$FRAMEWORK_EXECUTABLE_PATH"
mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"

done
```