Mobile SDK

Developer's Guide

Version: 3.4.0

Date: 2018-10-11

KOFAX

Table of Contents

Preface

This guide includes the information you need to successfully integrate Mobile SDK components into your mobile application.

For additional details on API library properties and settings, refer to the Mobile SDK API Reference Guide.

Technical Specifications document on the Kofax website at `http://services.kofax.com/support/products/mobile-sdk/3.x/supported-configurations.php`.

Getting help for Kofax products

Kofax regularly updates the Kofax Support site with the latest information about Kofax products.

To access some resources, you must have a valid Support Agreement with an authorized Kofax Reseller/Partner or with Kofax directly.

Use the tools that Kofax provides for researching and identifying issues. For example, use the Kofax Support site to search for answers about messages, keywords, and product issues. To access the Kofax Support page, go to www.kofax.com/support.

The Kofax Support page provides:

- Product information and release news

  Click a product family, select a product, and select a version number.
- Downloadable product documentation

  Click a product family, select a product, and click **Documentation**.
- Access to product knowledge bases

  Click **Knowledge Base**.
- Access to the Kofax Customer Portal (for eligible customers)

  Click **Account Management** and log in.

  To optimize your use of the portal, go to the Kofax Customer Portal login page and click the link to open the *Guide to the Kofax Support Portal*. This guide describes how to access the support site, what to do before contacting the support team, how to open a new case or view an open case, and what information to collect before opening a case.
- Access to support tools

  Click **Tools** and select the tool to use.
- Information about the support commitment for Kofax products

  Click **Support Details** and select **Kofax Support Commitment**.

Use these tools to find answers to questions that you have, to learn about new functionality, and to research possible solutions to current issues.

Product documentation

By default, the Mobile SDK documentation is available online. However, if necessary, you can also download the documentation to use offline.

Default online documentation

The product documentation for Mobile SDK 3.4.0 is available at the following location.

https://docshield.kofax.com/Portal/Products/en_US/KMC/3.4.0-o6num87075/SDK.htm

Configure offline documentation

To access the documentation offline, download KofaxMobileSDKDocumentation-3.4.0_EN.zip from the Kofax Fulfillment Site and extract it on a local drive available to your users.

The compressed file includes both `help` and `print` folders. The `print` folder contains all guides, such as the Installation Guide and the Administrator's Guide. The `help` folder contains APIs and other references.

Chapter 1

Introduction to the Mobile SDK

The Mobile SDK includes both image capture and image processing capabilities to integrate with applications designed to run on Android and iOS mobile devices.

The image capture features improve the quality of captured images, such as stability delay, camera orientation, flash support, and image frame guidelines.

The image processing features provide access to a patented processing technology that includes capabilities specific to images obtained from mobile device cameras.

Product overview

The Mobile SDK provides libraries, headers, code samples, documentation, and a help system that developers use to create mobile applications such as those that require image processing or connection to other systems for image capture.

This SDK is distributed as a zip file, which includes all development environments. The SDK libraries cannot be used without a license. Once you unzip the SDK, install the license within your own application or within the source code for the sample apps. The license key can be added either manually at run-time, or programmatically at build time.

The zip file contains several main folders:

Folder	Purpose
Android	Contains documentation, libraries and sample applications specifically for mobile devices using Android. The documentation folder contains a welcome page that launches the HTML help for the SDK API, including detailed information about all the SDK classes.
Hybrid	It contains three folders. Kofax PhoneGap Plugin: This folder contains Kofax PhoneGap Plugin,documentation, and sample applications. The documentation folder contains an `index.html` page that launches the HTML help for the plugin, including detailed information about all the plugin classes. HTML5 SDK: This folder contains HTML5 SDK, documentation and sample applications. The documentation folder contains an `index.html` page that launches the HTML help for the HTML5 SDK, including detailed information about all the HTML5 SDK classes.

iOS	Contains documentation, frameworks and sample apps specifically for Apple mobile devices running iOS. The documentation folder contains a welcome page that launches the HTML help for the SDK API, including detailed information about all the SDK classes. The welcome page includes links to more SDK details and provides information about the major grouping of SDK classes.
AppStats	Contains a sample Microsoft SQL Server create script for the App Statistics database schema.

Chapter 2

Conceptual overview

The Mobile SDK includes separate libraries to integrate with applications designed to run on Android and iOS mobile devices.

In addition to the image capture and processing functions already mentioned, the SDK also allows interface exchanges and integration with the Kofax Front Office Server (KFS) and Kofax TotalAgility servers.

All methods are synchronous, unless otherwise specified. For asynchronous methods, the application level is notified by library events when operations are complete. The event design is OS-specific. The iOS library uses delegates, and Android uses callback methods and associated indications. The application displays all error messages as needed. Most methods include an error value that can be accessed when the event fires to indicate why a method failed.

Note For Android, there is a class called `AppContextProvider` that must be initialized by the application before using the mobile SDK API.

```
/**
* This class keeps a reference of the application context.
* In order to use the Mobile SDK, you must first set
* a valid application context.
* Alternately, you can specify the AppContextProvider class
* in your application's Manifest, as an attribute of the
* <application/> element:
*
 *   <application \n
*       android:icon="@drawable/ic_launcher" \n
*       android:name="com.kofax.kmc.kut.utilities.AppContextProvider" \n
*       android:label="@string/app_name" \n
*       android:theme="@style/AppTheme" >
*   </application>
*/
```

Note For iOS, most class methods return error code enumeration values. The error enumerations are included in an error info header file. You can use methods in the error class to obtain a description of the error, and recommended corrective actions to prevent the error, if applicable.

The Mobile SDK is designed to make it simple for you to build mobile applications that can extract information from images of documents such as: checks, bills, and IDs such as driver licenses and passports. A typical use case that can be enabled with the Mobile SDK is to take a photograph of the front and back of a check and then click a button to deposit it into your bank account. This very use case is enabled using a combination of the Kofax Mobile Capture SDK and the Kofax Mobile Remote Deposit Capture™ framework.

Extracting information from an image using the SDK consists of three steps: (1) Capturing an image (2) Perfecting the image (3) Submitting it to one of the Kofax Mobile Frameworks such as: Kofax Mobile Bill Pay, Kofax Mobile Remote Deposit Capture, and Kofax Mobile ID Capture.

User recommendations for taking a photograph

While using the library to perform camera-based image processing, the results are dependent upon the quality of the original photograph. To ensure that users achieve optimal results, they should be encouraged to follow certain recommendations:

- When possible, set the camera resolution to a minimum of 5 MP or 8 MP for larger documents.
- Do not use zoom. If it is available, it must be set to 1x.
- Flatten wrinkled pages or upturned corners even if they do not include data.
- Place the document on a flat non-cluttered surface. This surface should have a distinct, relatively uniform background with as little variation as possible. Avoid backgrounds that look too much like the border areas of the document itself. Desk surface texture is OK, but sharp colors or brightness differences in the background cause problems for page detection.
- Avoid shadows.
- Check the lighting before taking a photo. Good uniform illumination will help to get a faster shot without motion blur and avoid jitter noise because of insufficient light.
- Avoid using flash which can over saturate the picture or wash out a part of the image.
- Maximize the area within the image frame occupied by the document, but make sure that there is a small margin of background surrounding the document. For a standard letter-size page this margin should be about 0.5", for documents of other sizes it should be proportionately smaller or larger.
- Rectangular overhead camera shots are best, but in order to avoid shadows cast by the camera itself it is OK to take the picture from an angle - resulting keystone distortions will be corrected. However, larger angles should be avoided - not because of larger keystone distortions (these can be corrected for most angles), but because of limited depth of field. The rule of thumb is that the depth of field is 27 mm (just over 1 inch) for a picture taken from a distance of 1 foot. So, if the difference between the distances to the most distant point and the closest point exceeds the depth of field, some parts of the document will be blurred.
- If available, use the touch focus feature to focus on the center of the document (or the center of the area of interest).

Capturing an image

The first step in accurately extracting data from an image is to take a high quality image. But what does high-quality mean in the context of data extraction? A high-quality image in this context must be well-focused, well-lit, well-centered, and have a high-resolution. This cannot be emphasized enough. The key to having a high rate of extraction accuracy is to start with a high-quality image.

General recommendations

As a general rule, do not attempt to capture documents that have been placed on a surface with complex patterns, shapes, or colors. A plain, contrasting surface is recommended. For best image capture results, start with the following suggested resolutions and if necessary make appropriate adjustments.

Focus

The image capture software on most phones has built in auto-focus capability. However some devices are better than others at maintaining focus.

In general, iPhone devices of all generations are very good at focus and Android devices vary widely.

The SDK contains tools (for example, Quick Analysis) that allow you to test an image after it has been captured to determine whether it is well-focused. Lighting and focus are related in that cameras generally take longer to focus in poor light and so an image taken in poor light is also more likely to be blurry.

Another problem that can impact focus and lead to blurry images is a lack of contrast on a page. An example of a document that can sometimes lack adequate contrast is the back of a check. There is usually very little machine-printed text on the back of a check and the only item on the check that might show some contrast is the endorsement. The Mobile Capture SDK includes a `SetFocusArea` method that allows an application to define the exact area that the camera should "look at" as it attempts to focus.

Capturing a well-lit image

It is best to capture an image in a well-lit area. Diffused light that does not cause a lot of glare is best. But if this is not possible, the SDK does allow you to turn on your smart phone's lamp if it is equipped with one.

Well-centered image

Most people are able to take a well-centered image, but the SDK allows you to show frames of various sizes and aspect ratios on the screen that guide the user to center the document in the view finder.

Undistorted image

If the device is not held parallel to the document, various types of perspective distortions will appear in the captured image. To help minimize these distortions the SDK allows you to show guidance on the screen that helps the user hold the camera parallel to the document.

Diagnostic information

Diagnostic information can be displayed at the top of the image capture screen for both iOS and Android. This includes the following information:

- If the image is focused.
- Stability status, including value and threshold.
- Camera resolution.
- If the camera is level.
- Pitch and roll, including the threshold for each.

Enable diagnostic information to appear before you begin the image capture. Do the following:

1. At the screen where you initiate the capture (such as ID Card), tap **Settings** in the upper right corner.
2. Tap **Camera Settings** (iOS) or **Camera** (Android).
3. Turn on **Diagnostics**.
4. Go back to the screen where you initiate the capture and tap the button to capture the image.
   The diagnostic information appears at the top of the screen.

Auto-Capture

There is a tension between usability (the desire of the user to be able to capture an image with a minimum amount of instruction) and the need to capture a good image. The goal of auto-capture is to guide the user to position the device to capture a well-centered and focused image that occupies the greatest possible area on the screen so the resolution of the document within the image (the dpi value) is maximized.

The SDK supports both auto-capture and manual capture. In manual capture, your application simply displays a guide on the screen indicating how the user should position the document and provides them with a button that they need to press to take the picture. Even if auto-capture is used, the capture experience will switch to manual capture if the user is not able to capture with a configurable length of time. In general, auto-capture is preferred and will lead to a better image most of the time.

Image processing recommendations

The SDK has a robust set of image perfection/processing features. This section provides an overview and explains the most commonly used image processing parameters.

In general it is impossible for the user to create a perfect image, and so the SDK provides an Image Processing object that provides tools that the user can use to "perfect" their images. The image processing object allows you to: crop, deskew, and binarize the image. In most cases (with the exception of IDs and driver licenses) you will always want to crop and deskew the image.

Because image processing can be slow on some devices you may also want to convert to gray scale first before performing additional processing. This will speed up processing on slower devices and is better than down-scaling (which is another way to improve performance on slower/older devices) because it does not incur any loss of data.

Typically devices that are the slowest also have the lowest resolution which means that down-scaling can be problematic, especially if one is capturing a larger document.

Following are some suggestions for modifying the image processing string.

- Document images will come from phone cameras: this warrants adding `_DeviceType_2`.
- Documents are letter-size - this allows adding `_DocDimLarge_11.0_DocDimSmall_8.5_`.
- Content is parallel to document edges: this warrants adding `_DoSkewCorrectionPage_` and `_DoCropCorrection_` in order to deskew and crop them.
- The images sent to OCR should be binarized: this means adding `_DoBinarization_`.
- As long as binarization is necessary it makes sense to auto-orient them - this means adding `_Do90DegreeRotation_4`.
- The text of interest is black against bright backgrounds: this warrants adding `_LoadSetting_<Property Name="CSkewDetect.convert_to_gray.Bool" Value="1" />_` because this will speed up processing without any degradation of binary images.
- Assuming reasonable lighting conditions, binarization should work fine even without normalization of illumination: this warrants adding `_LoadSetting_<Property Name="CSkewDetect.correct_illumination.Bool" Value = "0"/>"`; however, while this

setting will save processing time there is a slim chance of OCR degradation, so it may be useful to try with and without this setting

Submitting an image to the Kofax Mobile Frameworks

The Kofax Mobile Frameworks are a set of server-based business process components that run within the Kofax Transformation Module (KTM) and are exposed via a RESTFUL Web service (mobile and internet friendly) called the Real-Time Transformation Interface. The following business processes are offered: Kofax Mobile Bill Pay, Kofax Mobile Deposit Capture, Kofax Mobile ID Capture. Each of these products are explained below.

Kofax Mobile Bill Pay

The Kofax Mobile Bill Pay framework enables banks to better engage consumers via their mobile device, and by empowering customers to easily and effectively capture bills and add payees. Utilizing their smart phone or tablet, customers simply snap a picture of their bill, and then the data is extracted, corrected, and perfected by Kofax technology. Finally, the information is automatically presented to the user for easy bill payment.

These frameworks expose a RESTFUL API which is a Web service that is accessible via a URL and standard HTTP protocols. The frameworks return their data as JSON. The code to create an HTTP request sends it to one of the mobile frameworks and then processes the request, as shown below.

```
protected String doInBackground(String... arg0) {

  {
 Create an httpClient object
 Create a ResponseHandler object
 Convert the image you are passing to RTTI to an array of bytes
 Create a requestEntity object passing in the array of bytes

 Within the appropriate error handling
{
 Setup an HTTPPut or Post request
 Add parameters and a header
 Set the entity equal to the requestEntity object created above
 Call the execute method on the httpClient object passing in the BasicResponseHandler
 Receive and process the JSON response
}

   }
```

This pseudo code is based on Java.

Kofax Remote Deposit Capture

The Kofax Remote Deposit Capture framework provides banking customers with high-quality check image capture capabilities via Kofax software installed and executed on their mobile device to ensure images are acceptable for downstream processing.

This process is delivered quickly and easily without the need for specialized application developers, resulting in a faster ROI for financial institutions. Kofax technology captures all data elements in checks from numerous countries, and customers can make deposits from anywhere. Patented image perfection technology ensures the accurate capture of checks, with no manual entry required, resulting in a better user experience.

Kofax Mobile ID Capture

The Kofax Mobile ID Capture framework enables organizations to quickly and easily provide the ability for customers to take a picture of a driver license or other forms of ID, and have the information extracted and populated into a mobile application.

HTML 5 Capture

The mobile capture SDK is implemented as a native code library for iOS and Android. As such, it requires the end user to install an application on their device before they can use any of the functionality it offers. However there are some use cases where you may wish to leverage some of the functionality offered in the Kofax Mobile Capture platform without requiring the user to download and install an application on their mobile device. To support these use cases, the Mobile SDK includes tools that make it easy for you to build thin-client, HTML 5 applications.

Refer to the *Kofax MobilHTML5 SDK Developer's Guide* for details.

Chapter 3

Getting Started with the SDK

The following sections provide information on getting started with the Mobile SDK.

Setting up for localization

The SDK supports localization for 5 non-English languages: Brazilian Portugese, French, German, Italian and Spanish. The Mobile SDK is fully localized and supports the following configurations: iOS and Android. The following information describes the requirements to build a localized native Android application using the SDK. However, this information is not intended to provide a full tutorial on developing a localized application.

iOS

The strings required for a localized SDK are packaged in the SDKStrings.bundle file, which is delivered as part of the Mobile SDK release contents. This file must be included as part of the application project. If the application is then built for the non-English languages supported by the MobileSDK , SDK strings will be displayed in the chosen language. The application can then be localized independently from the SDK localization.

Android

In order to support localization, an Android application must create string resource files to contain the localized versions of strings. In order to build an application using the Mobile SDK, the SDK strings must be merged with the application resources. When using Android Studio, the SDK strings are included in the SDK AAR files, and the resource merge is automatic. If not using Android Studio, the SDK strings must be manually merged. The Android application can be localized without regard to the number or contents of SDK strings.

Using the SDK with iOS

This information assumes you are using XCode 8 to build a native application for iOS 8 or above.

1. Create a new iOS project.
2. Unzip and include the files from `MobileSDK.zip` in the `Frameworks` folder under “iOS”. These are:
   - `MobileSDK.framework`
   - `SDKStrings.bundle`
   - `uiImages.bundle`

In addition, you need to include the following native iOS frameworks and libraries:

- AVFoundation.framework
- CoreLocation.framework
- CoreMedia.framework
- SystemConfiguration.framework
- AudioToolbox.framework
- AssetsLibrary.framework
- CoreMotion.framework
- MobileCoreServices.framework
- Accelerate.framework
- libsqlite3.tbd
- libz.tbd
- libxml2.tbd
- libc++.tbd

Note The `uiimages.bundle` and `SDKStrings.bundle` specifically need to be included by choosing "Add Files" in the XCode's project navigator. Although XCode will appear to allow you to add it through "Link Binary With Libraries" on the build phases tab, that will silently fail.

Note For version of xCode prior to 7, use libsqlite3.dylib, libz.dylib, libxml2.dylib, and libc++.dylib instead of libsqlite3.tbd, libz.tbd, libxml2.tbd, and libc++.tbd

3. Include the license header file containing your valid SDK license, and import it in the top of your app's AppDelegate file, along with the Utilities framework as follows:

```
#import “kfxEVRS_License.h”
#import "kfxLibUtilities/kfxUtilities.h"
Somewhere in the initialization of your application, such as in your
 AppDelegate’s
“didFinishLaunchingWithOptions” method, make the following call:

kfxKUTLicensing *licenseConfig = [[kfxKUTLicensing alloc] init];
    if([licenseConfig setMobileSDKLicense: PROCESS_PAGE_SDK_LICENSE] ==
 KMC_SUCCESS)
{
 // License is valid
}
```

4. The application project file needs to specify the location of the `MobileSDK.framework/Headers` in "Header Search Paths."
5. If you are using XCode 7 / iOS 9 or later, and your application is intended to connect to a server, be sure to review Apple's documentation with regard to App Transport Security.
6. Starting iOS 10, in order to allow access to a camera, be sure to extend your application’s info.plist file with NSCameraUsageDescription. See Apple’s documentation in regards to Cocoa Keys.
7. Starting iOS 10, in order to allow access to the photo gallery, be sure to extend your application’s info.plist with NSPhotoLibraryUsageDescription. See Apple’s documentation in regards to Cocoa Keys.

8. In order to use Mobile SDK from Swift based project, be sure to update the bridging header file to import all required Mobile SDK API. See Apple's documents in regards to using Swift and Objective-C in the Same Project.
9. Swift sample applications are based on Swift 3.0. Use xCode 8 or higher in order to build them.

Now that the basic project has been set up, you can begin developing your application and incorporating other SDK elements.

Sample iOS projects

These sample projects guide you through the basic steps for capturing an image, reviewing and performing image processing operations, sending the image to a server for data extraction, and finally displaying the results. They are shipped with the product and can be found in `iOS\SampleApps\Native\SampleAppsX.X_iOS.zip`

The following sample applications are included with the Mobile SDK.

Note The sample apps are here, the Kofax Mobile Demo and the EasySnapApp are near the end of this document.

Sample Project	Description
`CheckAnimation`	Demonstrates capturing an image with the Check Experience. This sample shows how to initialize the `kfxKUIImageCaptureControl` and set up `kfxKUICheckCaptureExperience` with custom configurations, and display a captured image with `kfxKUIImageReviewAndEdit`.
`DocumentGuides`	Demonstrates capturing an image with the Document Experience. This sample shows how to initialize `kfxKUIImageCaptureControl` and set up `kfxKUIDocumentCaptureExperience` with custom configurations, and display a captured image with `kfxKUIImageReviewAndEdit`.
`ImageProcessor` `SwiftImageProcessor`	Demonstrates image processing with the perfection profile. This sample shows how to initialize and set up `kfxKENImageProcessor` along with `kfxKEDImagePerfectionProfile`, and display a processed image with `kfxKUIImageReviewAndEdit` .
`ServerAPI`	Demonstrates data extraction with the server API. This sample shows how to send a processed image to a server for data extraction and how to handle the response.

`OnDeviceExtractionSample`
`SwiftOnDeviceExtractionSample`
`ODECustomProviderSample`
	Demonstrates On-Device Extraction API for ID documents. The sample shows how to initialize and set up `kfxKOEIDExtractor` with server asset files and display extracted fields.

Sample Project	Description

`BillCaptureSample`
`CheckCaptureSample`
`SwiftCheckCaptureSample`
`CreditCardCaptureSample`
`IDCaptureSample`
`PassportCaptureSample`
	Demonstrates capturing, image processing, and extraction of corresponding documents with Packaged Capture Experience.

`PassportGuides` `SwiftPassportGuides`	Demonstrates capturing an image with the Passport Experience. This sample shows how to initialize the `kfxKUIImageCaptureControl` and set up `KFXPassportCaptureExperience` with custom configurations, and display a captured image with `kfxKUIImageReviewAndEdit`.
`IDVerification`	Demonstrates the mobile SDK's ID verification features integrated with On Device ID Extraction. See the *Kofax Mobile ID Verification Administrator's Guide* for more details.
`FixedAspectRatioGuides`	Demonstrates capturing an image with the Fixed Aspect Ratio Experience. This sample shows how to initialize `kfxKUIImageCaptureControl` and set up `KFXFixedAspectRatioCaptureExperience` with custom configurations, and display a captured image with `kfxKUIImageReviewAndEdit`.

The samples presented in the above table are recommended as a starting point for becoming familiar with using the Mobile SDK. These start from the bottom, where you will learn how to capture an image with the set of available experiences, display the captured image in the UI for review, process the image, and extract OCR data from a server.

Building iOS sample apps

The following steps are one-time setup to build all of the sample apps:

1. Unzip the SampleAppsx.x_iOS.zip file.
2. Inside the SampleAppsx.x_iOS folder, create a folder named KofaxFrameworks,
3. Copy the files from the SDK/iOS/Frameworks/MobileSDK.zip file into the KofaxFrameworks folder, i.e.: MobileSDK.framework, SDKStrings.bundle, amd uiimages.bundle.

These steps are general step for each sample app you wish to build:

1. Update app bundle ID and development team as needed for code signing
2. Search and replace the string "MyLicenseString" with your SDK license.

Additional steps are required for some of the sample apps:

`ODECustomProviderSample`: Unzip files from SDK/iOS/Frameworks/sdk-sources.zip into the sdk-sources folder.

And `ODECustomProviderSample` requires server extraction model files taken from the Kofax MobileID server product, for the regions you wish to extract IDs for:

Create an Assets folder within the project, as shown in ODECustomProviderSample. Unzip project files and variants into Assets directory: `Assets/USIDs/[project files, variant folders]`

Project.zip contents should extracted to the root of the USIDs folder and each `Variant.zip` to the root of a corresponding variant folder. So the resulting model structure should look like this:

```
Assets/
  USIDs/
    Classifier.config
    Classifier.model
    Fields.xml
    VariantsList.json
    cities.zip (optional)
    streets.zip (optional)
    AK1-AK1_c-AK1_id/
      EvrsOpString1.txt
      EvrsOpString2.txt
      Extractor.config
      Extractor.model
    AK2-AK2_id/
      EvrsOpString1.txt
      EvrsOpString2.txt
      Extractor.config
      Extractor.model
    ...
```

Set up licensing

The first step is to set up the licensing information for the Mobile SDK. Replace the licensing string in the `viewDidLoad` method in `CaptureViewController.m` with the one provided to you when you purchased the Mobile SDK.

When the capture view controller is loaded, the licensing information is included. The view controller sets itself as a delegate for getting messages from the `kfxKUIImageCaptureControl` object.

Image capture

After the `CaptureViewController` has been fully created, an instance of the appropriate capture experience (`kfxKUICheckCaptureExperience`, `kfxKUIDocumentCaptureExperience`, or `KFXPassportCaptureExperience`, depending on the sample) is created with `imageCaptureControl` and its criteria as parameters. The capture experience object provides an interface that sets up a set of frames and messages that will help guide the user.

For example, you may want to define a smaller frame for capturing smaller documents such as a business card or a credit card. The experience object also enforces constraints for taking the image, which are specified through the criteria object. Some examples of the constraints are document page detect, thresholds for stability, and pitch and roll. Only when the constraints are met will the picture be taken. There is a way to ignore the constraints and force the application to take a picture via a camera button in the toolbar.

Image review

Once the image has been taken, it will be available through the `imageCaptureControl:imageCaptured:` delegate method. The image is retrieved and passed to the `ReviewViewController`, which displays the object on the screen. This object also provides an interface to crop the image. The user can return to the capture screen by pressing the Close button.

The `ImageProcessor` and `ServerAPI` samples also provide functionality to pick a photo from the gallery and use it as a captured image for review. They also perform image processing operations in `viewDidLoad` from `ReviewVewController.m`.

Image processing

The captured image is provided to the image processing object along with the image perfection profile. The image perfection profile provides an option to apply advanced image processing options, which can be specified through a string or a file.

Once the image processing is completed, the processed image is provided through the delegate method `imageOut:withMsg:andOutputImage:`. Then, the processed image is passed on for review, instead of the captured image.

The `ServerAPI` sample passes a processed image to the Real-Time Transformation Interface server for data extraction.

Data extraction

The processed image is sent to the Real-Time Transformation Interface server over an HTTP connection. The Real-Time Transformation Interface server will have different URLs for specific document types.

For example, the URL used for a Bill Pay document will be different than the URL used for a Check Deposit type document. The Real-Time Transformation Interface server extracts the relevant data from the image.

For example, when a check image is sent, the Real-Time Transformation Interface server will extract check-specific information such as the routing number, account number, and amount. Similarly when a bill is sent, the Real-Time Transformation Interface server will extract bill specific information such as biller name, address, account number, and amount due.

Results

The Real-Time Transformation Interface server sends the response in JSON format. The sample application extracts the key/value pair from the JSON string and displays the result in a text view.

Using the SDK with Android

Use the following to set up the SDK for use with Android.

Setting up your project

1. Get the entire `MobileSDK_Libs/aar/` folder contents and copy that into the `libs/` folder of your project.
2. Add the `libs` folder as a flat file repository and the SDK reference by adding this to your project's `dependencies{}` closure.

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
compile(name:'sdk-release', ext:'aar')
```

3. If Android Studio is not used to build the application, merge the SDK strings and drawables with the application Resources.

Now the basic project has been set up, so you can begin developing your application and incorporating other SDK elements.

To use the optional libraries, follow the application build steps as shown, then remove the folders `arm64-v8a` and `armeabi-v7a` from the application `libs` folder. Copy the folders `x86` and `x86_64` from the `optional` folder in the SDK to the application `libs` folder to use the optional libraries.

Component names and descriptions for the Android SDK

Library Name	Description
sdk-release.aar	AAR encapsulates all SDK features: the capture feature with guidance, view controller for displaying an image, classes for bar code and credit card capturing, image processing, classification, and document detection, data extraction. It also contains utility classes for error handling, working with licensing and for AppStats etc.
card.io-5.5.1.2.jar	Library for Credit Card capturing and extraction. Not needed if app does not require Credit Card support.
commons-lang3-3.5.jar	Apache library provides helper utilities for the java.lang API. It is used by many features.
otto-1.3.8.jar	Library provides event bus functionality. Used by Image Capture View. Needed if application requires capture feature.
isg-3.3.0.0.0.673.jar	Library for image processing and edge detection. Used by capture, image classification and document detection features.
dagger-2.11.jar javax.inject-1.jar	Framework for Dependency Injection to Android. It is used by many features.
gson-2.8.1.jar	Library for work with JSON format and used by Extraction feature.
bolts-android-1.4.0.jar bolts-applinks-1.4.0.jar bolts-tasks-1.4.0.jar	Library for work with asynchronous tasks used by many features.
commons-io-2.5.jar	Library used for file copying by On-Device Extraction feature and SDK sample apps.

Library Name	Description
retrofit-2.3.0.jar converter-gson-2.3.0.jar okhttp-3.8.0.jar okio-1.13.0.jar	Libraries used for network connection by On-Device Extraction feature.
mobilebarcodeparser-3.3.0.0.0.39.jar	Library for On Device Extraction feature only.
manatee-2.5.0.0.0.41.jar	Library for Barcode capture, used in many features: BarcodeCaptureVIew, On-Device Extraction, License capture.
common extractionengine -3.3.0.0.0.75.jar	OCR engine wrapper library for On-Device Extraction feature.
xvrs-3.3.0.0.0.106.jar	New image processing and cropping library. Required for Fixed aspect ratio experience and Image processor.
tesseract-4.0.0.5.jar	Wrapper library for Tesseract OCR engine used for non-Latin characters by the On-Device Extraction feature.

Name	armeabi-v7a (v 3.2)	arm64-v8a (v 3.2)	armeabi-v7a (v 3.3)	arm64-v8a (v 3.3)	Description
libgnustl_shared.so	694K	1.0M	698K	1.0M	The GNU Standard C ++ Library. It is used by libraries for image processing and document detection.
libsol_isg_mobile.so	1.9M	3.8M	1.9M	3.8M	The ISG library. Used for capturing, image clarification and document detection features.
libdetection_based_tracker.so	21K	37K	21K	41K	Library used for image capturing based on GPU algorithm, clarification and document detection features.
libopencv_java3.so	4.4M	7.4M	6.6M	12M	The Open CV library. Used for image clarification and document detection features.
libEvrsJniWrapper.so libKfxEVRS.so	8.5M	8.8M	8.9M	9.2M	Library for image processing, SDK versions and license management. Used by the ISG library for document detection.

Name	armeabi-v7a (v 3.2)	arm64-v8a (v 3.2)	armeabi-v7a (v 3.3)	arm64-v8a (v 3.3)	Description
libAtalaBar.so	214K	363K	214K	363K	Used for performing bar code recognition by BarCodeReader and On Device Extraction features.
libKfxBarcodeScannerLib.so	257K	411K	257K	411K	The Barcode Scanner library is a package of bar code detection and decoding procedures. Used for bar code capture and detection features.
libbarcode_parser.so	650K	1.1M	286K	727K	Additional Kofax library for On-Device Extraction feature (barcode parsing)
libcardioDecider.so libcardioRecognizer.so libcardioRecognizer_tegra2.so	1.5M	2M	1.6M	2.1M	Libraries used for Credit Card extraction feature. Not needed if app does not require Credit Card support. Note To be clear: These libraries do not support non- embossed cards.
libcommonextractionengine.so	-	-	93K	214K	OCR engine wrapper library for On-Device Extraction feature.
libXVrs.so	-	-	926K	1.7M	New image processing and cropping library. Required for Fixed aspect ratio experience and Image processor.
liblept.so	-	-	1.5M	2.6M	Image processing library used by Tesseract OCR engine.
libtess.so	-	-	1.8M	3.1M	Tesseract OCR engine library.

Removing optional libraries

If desired, the Mobile SDK can be used without some libraries if corresponding functionality is not required.

For example, the user may delete the Card IO native libraries (libcardioRecognizer.so, libcardioRecognizer_tegra2.so, libcardioDecider.so) and card.io-x.x.x.x.jar if the Credit Card Extraction feature is not used.

Also, the user may delete the libAtalaBar.so library if the `BarCodeReader` API and On-Device Extraction (ODE) features are not used.

The user may also choose to delete the libBarcodeScannerLib.so library if the `BarCodeCaptureView` `LicenseCaptureView` APIs, and On-Device Extraction features are not used.

If non-Latin characters extraction for On-Device Extraction feature is not required, tesseract-x.x.x.x.jar, libtess.so, and liblept.so can be deleted.

Note KofaxMobileDemo app is not updated to have extra UI notifications, in case it is compiled without the libAtalabar.so\libBarcodeReaderLib.so\libcardio*.so libraries.

Android APK split mechanism

Google supports publishing multiple APKs for a single application that are each targeted to different device configurations. This is the best way to reduce the size of your application when using the Mobile SDK. The final application size will decrease by approximately 15 - 18 MB (depending on architecture).

For example, if the universal APK is 27.4 MB, the sizes of the split APKs will be the following: 9.4 MB (armeabi), 11.4 MB (arm64-v8a), 11.8 MB (armeabi-v7a)

To enable the ABIs Splits mechanism, you should extend your gradle configuration with the following splits{} closure inside the android{} closure in your build.gradle file:

```
android {
  ...
  splits {
    abi {
      enable true
      reset()
      include 'armeabi', 'armeabi-v7a', 'arm64-v8a'
      universalApk true
    }
  }
}
```

Sample Android projects

These sample projects guide you through the basic steps for capturing an image, reviewing and performing image processing operations, sending the image to a server for data extraction, and finally displaying the results. They are shipped with the product and can be found in `Android\SampleApps\Native\SampleAppsX.X_Android.zip`.

The following sample applications are included with the Mobile SDK.

Note The sample apps are here, the Kofax Mobile Demo and the EasySnapApp are near the end of this document.

Sample Project	Description
`CheckAnimation`	Demonstrates capturing an image with the Check Experience. This sample shows how to initialize the `ImageCaptureView` and set up `CheckCaptureExperience` with custom configurations, and display a captured image with `ImgReviewEditCntrl`.

Sample Project	Description
`DocumentGuides`	Demonstrates capturing an image with the document experience. This sample shows how to initialize `ImageCaptureView` and set up `DocumentCaptureExperience` with custom configurations, and display a captured image with `ImgReviewEditCntrl`.
`ImageProcessor`	Demonstrates image processing with the perfection profile. This sample shows how to initialize and set up `ImageProcessor` along with `ImagePerfectionProfile`, and display a processed image with `ImgReviewEditCntrl` .
`ServerAPI`	Demonstrates data extraction with the server API. This sample shows how to send a processed image to a server for data extraction and how to handle the response.
`OnDeviceExtraction`	Demonstrates the On-Device Extraction API for ID documents. The sample shows how to initialize and set up `OnDeviceIdExtractor` with local asset files, and how to display extracted fields.
`ODEOverride`	Demonstrates how to customize On-Device Extraction. This sample shows how to adapt the project providers, and implement a cache.
`PassportGuides`	Demonstrates capturing an image with the passport experience. This sample shows how to initialize `ImageCaptureView` and set up `PassportCaptureExperience` with custom configurations, and display a captured image with ImgReviewEditCntrl.
`IDVerification`	Demonstrates the mobile SDK's ID verification features integrated with On Device ID Extraction. See the *Kofax Mobile ID Verification Administrator's Guide* for more details.
`FixedAspectRatioGuides`	Demonstrates capturing an image with the Fixed Aspect Ratio experience. This sample shows how to initialize `ImageCaptureView` and set up `FixedAspectRatioCaptureExperience` with custom configurations, and display a captured image with `ImgReviewEditCntrl`.

The samples presented in the above table are recommended as a starting point for becoming familiar with using the Mobile SDK. These start from the bottom, where you will learn how to capture an image with the set of available experiences, display the captured image in the UI for review, process the image, and extract OCR data from a server.

Note For details about setup of the SampleApps project in Android Studio please see the `README.txt` file in the root of `SampleAppsx.x_Android.zip`.

Set up licensing

The first step is to set up the licensing info for the Mobile SDK. Replace the licensing string in the `onCreate` method in the main activity of the sample with the one provided to you when you purchased the SDK.

Image capture

After the `ImageCaptureView` has been fully created, an instance of the appropriate capture experience (`CheckCaptureExperience`, `DocumentCaptureExperience`, or `PassportCaptureExperience`, depending on the sample) is created with `imageCaptureControl` and its criteria as parameters. The capture experience object provides an interface that sets up a set of frames and messages that will help guide the user.

For example, you may want to define a smaller frame for capturing smaller documents such as a business card or a credit card. The experience object also enforces constraints for taking the image, which are specified through the criteria object. Some examples of the constraints are document page detection, thresholds for stability, and pitch and roll. Only when the constraints are met will the picture be taken. There is a way to ignore the constraints and force the application to take a picture via a camera button in the UI.

Image review

Once the image has been taken, it will be available through the listener that was set up with the `addOnImageCapturedListener` method from `ImageCaptureControl`.

The image is retrieved and passed to the `ImgReviewEditCntrl`, which displays the object on the screen. This object also provides an interface to crop the image. The user can return to the capture screen by pressing the Back button.

The `ImageProcessor` and `ServerAPI` samples also provide functionality to pick a photo from the gallery and use it as a captured image for review. They also perform image processing operations upon pressing the Process button.

Image processing

The captured image is provided to the image processing object along with the image perfection profile. The image perfection profile provides an option to apply advanced image processing options, which can be specified through a string or a file.

Once the image processing is completed, the processed image is provided to the `imageOut` method of the listener that was set up with the `addImageOutEventListener` from `ImageProcessor`.

Then, the processed image is passed on for review, instead of the captured image.

The `ServerAPI` sample passes a processed image to the Real-Time Transformation Interface server for data extraction upon pressing the Send button.

Data extraction

The processed image is sent to the Real-Time Transformation Interface server over an HTTP connection. The Real-Time Transformation Interface server will have different URLs for specific document types.

For example, the URL used for a Bill Pay document will be different than the URL used for a Check Deposit type document. The Real-Time Transformation Interface server extracts the relevant data from the image.

For example, when a check image is sent, the Real-Time Transformation Interface server will extract check-specific information such as the routing number, account number, and amount. Similarly when a bill is sent, the Real-Time Transformation Interface server will extract bill specific information such as biller name, address, account number, and amount due.

Results

The Real-Time Transformation Interface server sends the response in JSON format. The sample application extracts the key/value pair from the JSON string and displays the result in a text view.

Obfuscating applications with ProGuard

Here are our recommendations for obfuscating SDK apps using ProGuard . The following options need to be added to a ProGuard config file:

```
# Kofax SDK specific rules
-dontwarn java.nio.**
-dontwarn org.codehaus.mojo.**
-dontwarn java.lang.reflect.**
-dontwarn java.lang.invoke.**
-dontwarn com.kofax.**
-keep class com.kofax.** { *; }
-keep enum com.kofax.** { *; }
-keep interface com.kofax.** { *; }
-dontwarn io.card.**
-keep class io.card.** { *; }

# Gson specific rules
-keep class sun.misc.Unsafe { *; }
-keep class com.google.gson.examples.android.model.** { *; }
-keep class * implements com.google.gson.TypeAdapterFactory
-keep class * implements com.google.gson.JsonSerializer
-keep class * implements com.google.gson.JsonDeserializer

# OkHttp3 specific rules
-dontwarn okhttp3.**
-dontwarn okio.**
-keep class okhttp3.** { *; }
-keep interface okhttp3.* { *; }
```

Chapter 4

An In-Depth Look at the SDK

The following sections provide an in-depth look at the structure, organization, and capabilities of the SDK. For details on the classes, methods, parameters, and so on, refer to the reference guide that ships with the SDK.

Native interface object types

The native interface uses the following object types:

- Capture objects
- UI Control objects
- Engine objects
- Logistic objects
- Utility objects

Capture objects

Capture objects are containers for settings, preferences, results, and image data associated with image capture and related data (such as index field or bar code data).

UI control objects

UI Control objects are the only objects in the library that display a user interface screen, and they all involve an image display, either static or dynamic. These objects support capturing images, reviewing images, or capturing bar code images and data.

UI Control objects do not represent an entire screen, but instead are intended as a view contained within an application user interface screen. As such, these sizable objects can be controlled by other UI elements (for example, buttons, sliders), which are external to the UI Control objects and provided by the application.

Engine objects

Engine objects exist for image processing and classification. Each Engine object takes a single image and outputs images or associated data.

Logistics objects

Logistics objects represent a means of accessing raw input resources from an external source (such as a server) and submitting finished resources back to the same external source. An example would be downloading document types through a Logistics object, and then using one of them to create a document, add images and index fields to it, and then submit the finished document back to the server.

Note The logistics library can be used to connect to a Kofax TotalAgility server in the Kofax Azure environment so you can, for example, retrieve document types, submit images, login, and so forth

Utility objects

These are miscellaneous objects that don't belong in the other object categories. Utility objects include version objects to find the software version of individual libraries. You also use a License object to set your usage license.

Capturing images overview

In order to add an image to a document, it is necessary to use a previously initialized Document object. However, you can use capture images without associating those images with a document.

The Image Capture UI Control object provides an optimized means of capturing documents for subsequent image processing operations. You can also pass in feedback settings so that the Page Capture process displays an `ImageCaptureFrame` in a preferred orientation. The feedback helps ensure quality pictures with the native camera. This object returns only unprocessed images.

When you call the `takePicture` method on the Image Capture object, the library will display a view image screen of the specified size, in a rectangle of sufficient size to capture the image and still leave room for buttons you may want in the application.

The library will capture the image and return the Image object with an event when all the enabled feedback constraints are met, such as the camera is stable enough, level enough and oriented correctly. Upon getting the Image Taken event, the application can associate that image with any page of any Document object. The application must manage the images in a document. The returned image is an unprocessed raw image.

Library feedback and controls

When the library first opens the view, it displays an image capture frame that is designed to represent the form factor of the targeted document type. For instance, if the document type is for a 5 by 7 picture, the frame would have the same form factor, with the longest side along the longest side of the view rectangle. This gives the user a visual cue of how to frame the picture. The library uses the Image Capture Frame property of the Document type object associated with the Document object that is passed into Image Capture.

The application can set the color and line attributes of this frame.

The application can also specify landscape or portrait orientation, and a threshold to specify how close the image must be to that orientation. If the orientation is within this tolerance, the picture is allowed to be taken. If the preferred orientation is portrait, the longest side of the document type frame is shown

on the shortest side of the view rectangle, to indicate that the user should hold the device in the portrait orientation. If the preferred orientation is landscape, then the longest side of the reference frame is shown on the longest side of the view rectangle, with the frame rectangle aspect ratio expanded to fit within the view rectangle.

Also, the application can specify a stability setting such that the higher the value between 1 and 100, the more stable the camera must be held to take a picture.

Another type of feedback is camera levelness. The application can specify how level the camera must be in relation to the work surface to help user to keep the camera at a certain angle when taking a picture. You can set a threshold that specifies a plus/minus range around the preferred angle. For instance if all pictures are taken from a drafting table at +30 degrees, the application could set the preferred angle to +30, and the tolerance to +/- 5 degrees. If the pitch is within this tolerance, the picture can be taken.

Levelness is measured on two axes, pitch and roll. Pitch is rotation about the short axis, roll is rotation about the long axis, regardless of the orientation of the device.

Page detection

When using Page Detection, there is a noticeable amount of space needed around a sheet in order for page detection to work properly (find the page). You can use the image frame to help ensure that the user leaves an adequate margin.

If the frame is sized to be somewhat smaller than the image viewer, when capturing an image the user will then naturally attempt to fit the sheet within the frame, thus helping to ensure an adequate margin around the edges of the sheet.

Note This only applies when using page detection. If `pageDetect` is set to `False` (iOS) or the `setPageDetectMode(PageDetectMode.OFF)` method is called (Android), this margin is not required.

Image Capture Control object

Can be found in UIControls.

The Image Capture Control object is a smart camera view that takes pictures in single-shot or continuous mode. Features include user feedback cues to control levelness, camera shake, and framing. In continuous mode, users can take pictures of multiple documents automatically without explicitly tapping a "take photo" button in the application. Other properties of this object set the location and size of the control within the app's user interface. All the UI controls necessary to activate methods on the Image Capture control are provided by the application and are not part of the SDK. Gestures are provided and available to the application through native device platform events documented in the Android and iOS system API guides. The application is responsible for creating and displaying any screen overlays that offer user feedback, such as "hold camera still."

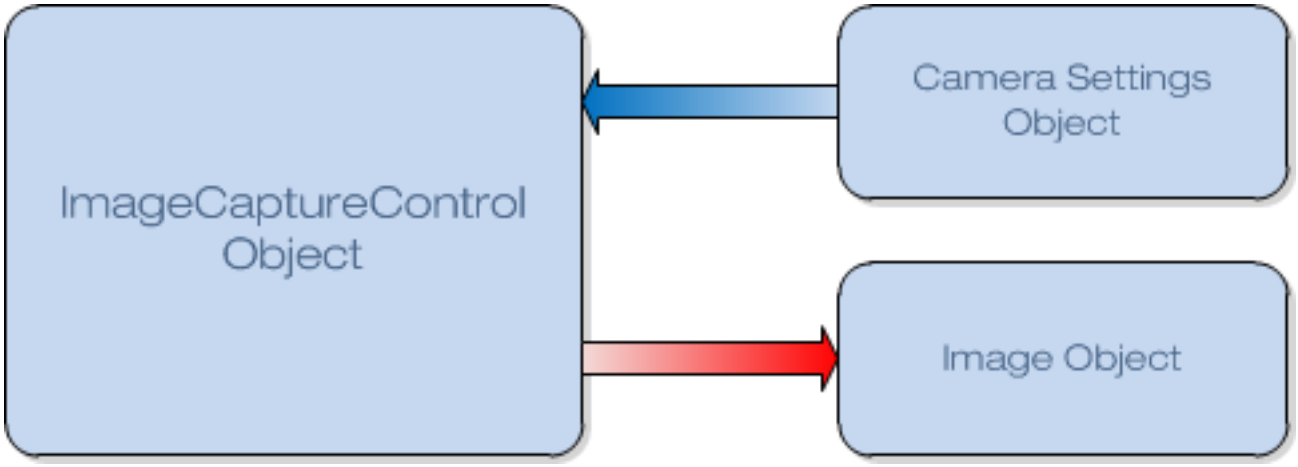


Image Capture Control Object Diagram

Set camera resolution

Camera resolution can be set automatically by the SDK, or the application can choose from a list of supported resolutions. This may be useful if the mobile devices has a higher resolution than needed. By selecting a lower resolution the application can conserve memory, and process the image faster. Additionally, bandwidth use will be reduced if the image is transferred via WiFi or the cellular network.

This is controlled by the `Resolution` property in the Image Capture Control. The default value for this property is `automatic`.

Camera LED lamp

The Image Capture View includes an API for controlling the camera's lamp. This allows the user of a device to more easily capture a quality image in low light conditions.

When the lamp is used to constantly emit light, it is referred to as a torch. This is in contrast to the momentary intense light emitted by a standard camera flash. The torch will be set through the existing flash properties on the Image Capture View by using a torch value in the corresponding enumerations. If the torch is unsupported on a particular device, an appropriate error will be raised.

For iOS
Use the flash `kfxKUIFlashSetting` enumeration.

```
typedef enum {
    ...
    kfxKUITorch = 3
} kfxKUIFlashSetting;
```

For Android
Use the Flash enumeration

```
public enum Flash {
   ...
   TORCH
}
```

Note On some Android devices, when the torch is used, the device is slow to perform exposure adjustments. If the image is taken before the exposure is set, it will appear washed out. This is particularly true when moving from a dark to a light area with continuous capture enabled.

Auto-Torch

The Auto-Torch feature forces the camera to automatically engage the flash LED when the detected luminance levels are too low. Therefore, if the user is pointing the camera at a poorly lit document, it will detect that it needs to turn on the torch. This improves the odds of capturing a good image even when lighting conditions are poor. The torch must be manually turned off.

The `Flash` enumeration includes an "AutoTorch" option that can be set in the `ImageCaptureView`.

Note If the camera does not support Flash, an exception will be thrown.

Get/Set focus area

The Image Capture View includes an API for controlling the size and location of the area in the view used for focusing the camera.

Device cameras typically focus by examining a small area of the image preview for contrasting detail. By default, this area is usually in the center of the preview.

If there is insufficient contrasting detail in this center, then the camera is unable to accurately focus, or in the worst case even be able to focus at all. Since some capture scenarios will result in a blank area in the default focus area, the developer needs to be able to specify an alternate focus area within the view.

Because the underlying capabilities of iOS and Android are very different, the Image Capture View exposes different APIs for them. The following API properties are included:

- `defaultFocusPoint` (iOS)
- `canSetFocusPoint` (iOS)
- `focusPoint` (iOS)
- `maxFocusAreas` (Android)
- `focusAreas` (Android)

For iOS

```
@interface kfxKUIImageCaptureControl : NSObject
 @property (readonly) CGPoint defaultFocusPoint;
 @property (readonly) BOOL canSetFocusPoint;
 @property (nonatomic) CGPoint focusPoint;
@end
```

For Android

```
public class ImageCaptureView extends RelativeLayout {
 public int getMaxFocusAreas ();
 public List<Rect> getFocusAreas ();
 public void setFocusAreas (List<Rect> areas);
}
```

Real-Time video feed

`PreviewFrameEvent` gives the developer access to video frames in near real-time. This makes it possible to perform application level logic on the contents of the video frame such as OCR or MICR data processing on each frame. The event is fired when a preview frame is ready from the camera. The event data includes the image data, and enough information to decode the data as a bitmap.

The real-time video feed capability also supports the following:

- **CheckDetector class**: This class, in the `Engines` suite of classes, is used to perform check detection. The class accepts an image and returns a `CheckDetectionResult` object with properties for the following:
  - Original image
  - Enum indicating check detection status (values for `None`, `CheckFrontDetected`, and `CheckBackDetected`)
  - Coordinates of the detected check in the original image
  - Subsampled image containing only the detected check image

  The returned object also contain check capture guidance: zoom, horizontal/vertical movement, and rotation.
- **MICR parser**: This class is in the `Utilities` suite of classes that extracts segments from a check MICR after check detection. The class accepts MICR extraction results from one of the algorithms used for check detection, and conditionally extracts the routing number, account number, and check number. A method in the `Image` object searches the image processing metadata for MICR data obtained by processing a check image. It returns the MICR data line as a string. MICR detection only applies to the front side of the check.

Check Capture Experience

The `CheckCaptureExperience` is an interface that provides user guidance to aid the user in capturing a quality image of a check. During use, the user is guided to take a picture of a check with a series of text messages that pop up on the screen. There are several properties to allow the user to customize the experience, including:

- Capture messages
  - `userInstruction`: Displayed when the camera is first launched, instructing the user on how best to capture the check.
  - `holdSteadyMessage`: Displayed to the user to hold the device steady if the device is almost ready to capture an image.
  - `centerMessage`: Displayed when the check is off center in the view finder.
  - `zoomOutMessage`: Displayed to the user if the check is too close to the camera.
  - `zoomInMessage`: Displayed to the user if the check is too far from the camera.
  - `capturedMessage`: Displayed to the user after a check is successfully captured.
  - `holdParallelMessage`: Displayed to the user when the device is not held level.
  - `rotateMessage`: Displayed to the user when the check is rotated in the view finder.

  There are also several properties that allow the user to customize the appearance of each capture message.

- Booleans
  - `vibrationEnabled`: A settable property to enable or disable haptic feedback when an image has been captured.
  - `tutorialEnabled`: A settable property to enable or disable the capture demonstration (visual demo that displays where the check should ideally be placed in the screen).
- Images
  - `tutorialSampleImage`: The sample image displayed when the capture demonstration is enabled
- Colors
  - `outerViewFinderColor`: a property that sets the color of the outer rectangle (the border). Essentially, this is the part that isn't the check.
- Other Properties
  - `checkSide`: A settable property to enable detection of either the front or back side of a check.
  - `checkDetectionSettings`: Contains the criteria that must be met for the detected check to be captured. The user can change these settings to make the check easier or harder to capture.

Note Devices that do not support 1080p or greater should use the Document Capture Experience for checks instead of the Check Capture Experience. The Check Capture Experience requires 1080p or greater for accurate check detection.

Document Capture Experience

The Document Capture Experience is an interface that displays messages to guide the user to capture a quality image of a document. It is designed to behave just like the Check Capture Experience, but is generalized to work for many document types.

The Document Capture Experience has most of the same properties as the Check Capture Experience, and can be configured identically. In addition, the Document Capture Experience includes the `longAxisThreshold` and `shortAxisThreshold`, which can be used to further specify how the document should fit within the frame and the Document Capture Experience.

The `DocumentCaptureExperience` is a subclass of the `DocumentBaseCaptureExperience`, which is also the superclass of the `CheckCaptureExperience`.

Passport Capture Experience

The Passport Capture Experience is an interface that displays messages to guide the user to capture a quality image of a passport. It is designed to behave just like the Check and Document Capture Experience, but is based on the real-time extraction of MRZ (Machine Readable Zone) lines.

The Passport Capture Experience has most of the same properties as Check Capture Experience, and can be configured identically. In addition, the Passport Capture Experience includes `aspectRatioTolerance`, which can be used to specify an acceptable window for aspect ratio of the passport.

The `PassportCaptureExperience` is a subclass of the `DocumentBaseCaptureExperience`.

Note We do not recommend using the torch or flash when capturing passports, since passports usually have laminated pages. The developer needs to set the Mobile SDK autotorch and flash APIs to OFF while using the Passport Capture Experience.

Fixed Aspect Ratio Capture Experience

The `FixedAspectRatioCaptureExperience` is an interface that displays messages to guide the user to capture a quality image of any document with a known aspect ratio. It will provide "Hold Steady" guidance only when the aspect ratio of the document and target frame are defined, such as when capturing credit cards or ID cards. The document is detected when it matches the target frame, or its sides, with a defined tolerance. The target frame sides are highlighted when it matches the document edges to provide intermediate user guidance. Once the document is detected (all four target frame sides are highlighted and document corners are found) the "Hold Steady" guidance will be displayed. This also includes the ability to take a picture of any document by holding it in hand until document edges are strong enough for detection.

The Fixed Aspect Ratio Capture Experience has most of the same properties as the Document Capture Experience, and can be configured identically. The exception is that the following messages are not supported for this experience: `centerMessage`, `zoomOutMessage`, `zoomInMessage`, and `rotateMessage`. By default all guidance messages are set to portrait orientation.

The `FixedAspectRatioCaptureExperience` is a subclass of the `DocumentBaseCaptureExperience`.

Selfie Capture Experience

The Selfie Capture Experience is an interface that displays messages to guide the user to take a intelligible Selfie. It is designed to perform a liveness check by looking for eye blinks.

During use, the user is guided by a series of text messages that pop up on the screen. There are several properties that allow the user to customize the experience, including:

- `UserInstructionMessage`: Displayed when the camera is first launched, instructing the user on how best to capture the selfie.
- `BlinkMessage`: Displayed after the user properly aligns his face in the target frame, instructing the user to blink his eyes.
- `capturedMessage`: Displayed to the user after a selfie is successfully captured.
- `outerViewFinderColor`: a property that sets the color of the target frame outer view.
- `frameColor`: a property that sets the color of the target frame border.

The Selfie Capture Experience has configurable selfie detection properties, which include `MinimumFaceSize`, `CenterPoint`, `TargetFrameAspectRatio`, `PaddingPercent` and `FaceAngleTolerance`. The `SelfieCaptureExperience` is a subclass of the `SelfieBaseCaptureExperience`.

Note the following:

- The Selfie Capture Experience has a dependency on the Google Play services vision API. Hence the application needs to set up a project with the Google Play services vision SDK, which downloads/ updates the required face detection packages during application installation time.

- When the app is installed on the device, Google Mobile Services (GMS) downloads a native library to perform detection. If the library has not finished downloading when the user attempts to use the facial detection feature, a message appears. When the download is completed, the user can then use the feature.
- The recommended minimum version is com.google.android.gms:play-services-vision:10.0.1. The device should have only a stable version of the Google Play services.

Packaged Capture Experience

A suite of Activities (Android) and View-Controllers (iOS) is exposed as part of the SDK to aid with the development of end-to-end capture experiences. The SDK API design is based on the Model-View-Controller pattern.

An application passes control of the entire screen while invoking the appropriate Capture View-Controller. The capture View-Controller encapsulates image capture, including user interface elements and settings specific to image processing and data extraction for each situation.

Once the image is captured, processed, and extracted, the result will be returned to the calling application as an appropriate Data Model class .

To configure the capture experience before starting it, an application needs to use a Parameters object from the Model. For example, setting the front or back of the check before calling the `CheckCaptureViewController` should be done using the `CheckCaptureParameters` class.

Note For iPhone, the Packaged Capture Experience is only supported on iOS 9 and above.

Activities and view controllers

This experience is based on the Model-View-Controller pattern. The following View- Controllers (Activities) implement the necessary capture experiences:

iOS

- kfxCheckCaptureViewController
- kfxBillCaptureViewController
- kfxIdCaptureViewController
- kfxPassportCaptureViewController
- kfxCreditCardViewController

Android

- CheckWorkflowActivity
- BillWorkflowActivity
- IdWorkflowActivity
- CreditCardWorkflowActivity
- PassportWorkflowActivity

The application will pass control of the entire screen while invoking the appropriate Capture View-Controller. The capture View-Controller encapsulates image capture, as well as UI elements and settings specific to image processing and data extraction for each situation.

Once the image has been captured, processed, and extracted, the result will be returned as an appropriate Data Model class to the calling application. To configure the capture experience before starting it, the application needs to use a Parameters object from the Model. For example, setting the front or back of the check before calling `CheckCaptureViewController` should be done using the `CheckCaptureParameters` class.

When referencing the JAR file format from the Kofax distribution (as opposed to the AAR file format), you will need to add a number of Activity declarations to the <Application> node in your application's AndroidManifest.xml before using the Packaged Capture Experiences.

Also, because the views associated with these Activities are designed to work in a specific orientation, do not change the extra orientation attributes in the XML, as doing so will affect their performance. The activity declarations are as follows:

```
<activity
  android:name="com.kofax.mobile.sdk.capture.check.CheckExtractActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.check.CheckCaptureActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.check.CheckWorkflowActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.ImageReviewActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="portrait" />
<activity
  android:name="com.kofax.mobile.sdk.capture.ProcessActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.bill.BillExtractActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.bill.BillCaptureActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.bill.BillWorkflowActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.id.IdExtractActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.id.IdCaptureActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.id.IdCaptureBackActivity"
```

```
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.id.IdWorkflowActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.passport.PassportExtractActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.passport.PassportCaptureActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.passport.PassportWorkflowActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="com.kofax.mobile.sdk.capture.creditcard.CreditCardWorkflowActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:screenOrientation="landscape" />
<activity
  android:name="io.card.payment.CardIOActivity"
  android:configChanges="orientation" />
<activity
  android:name="com.kofax.mobile.sdk.capture.creditcard.CardIoWrapperActivity" />
```

Note When using the Android Package Capture Experience to capture documents, the default implementation for the `IImageStorage` -- used to transfer bitmaps between activities -- is the `ContextImageStorage`. `ContextImageStorage` uses a static `HashMap` to store images by ID. It's important that captured images are removed and recycled by your application as soon as you are done using them, otherwise memory leaks will occur. Removing images and recycling can be done by calling `removeImage(String id)`; on the `IImageStorage` instance with the ID of the image you'd like to remove. Subsequently calling recycle() on the returned bitmap is required to release the bitmap data in memory immediately. Once all references to the bitmap are cleared, the bitmap will be eligible for garbage collection, avoiding leaks.

Note To be clear: io.card.payment.CardIOActivity does not support non-embossed cards.

Model classes

iOS

- `kfxCaptureData` (Image data includes captured image and processed image)
- `kfxCheckData` : `kfxCaptureData`
  - `kfxCheckIQAData`
  - `kfxCheckUsabilityData`
- `kfxBillData : kfxCaptureData`
- `kfxIDData : kfxCaptureData`
- `kfxPassportData : kfxCaptureData`
- `kfxCreditCardData : kfxCaptureData`

Android

- `Bill`
- `Check`
  - `CheckIQAData`
  - `CheckUsabilityData`
- `ID`
- `Passport`
- `IImageStorage` for captured and processed images

Capturing both sides of a document

Some documents have important information on both sides, such as checks and IDs. In such cases, both sides of the document may need to be captured. For improved accuracy, data extraction should be performed for both sides during the same session. To support this case, the corresponding document parameters will expose an API to set up processing an image for the other side of the document.

1. Set up a document capture ViewController/Activity for one side with extraction turned off.
2. Store a processed image with document data that will be available when the ViewController/Activity delegate/listener gets called.
3. Set up a second document capture ViewController/Activity for the other side of the document with extraction enabled. Set the appropriate image processing parameters for each image and process both of them.

Stability delay

The application can specify a stability threshold between 0 and 100. The higher the threshold, the more steady the camera must be held. If phone jitter is less than or equal to the stability threshold, the picture can be taken. This has the effect of delaying the image capture until the camera is sufficiently stable.

A value of zero turns the stability delay function off.

Once the threshold is set, the application calls `enableEvents`, which returns events with raw information about camera stability (as well as position and orientation). When all criteria are met, the application can call the takePicture method.

If the thresholds are easily met, the library takes a picture and sends an image captured event almost immediately.

To give the library more control over taking an image, set the thresholds to a narrower range. For example, if the stability threshold is set to 100, the image cannot be taken until the camera is perfectly steady. The library monitors all enabled feedback thresholds and does not allow the image to be taken until all criteria are satisfied.

Optionally, the application may still enable the feedback events, so that text cues ("such as hold the camera still ") can be presented to the user. The cues disappear when the feedback events indicate the criteria are met.

The application specifies how often (such as every 200 milliseconds) the library generates feedback events.

iOS

```
@interface kfxKUICaptureExperienceCriteriaHolder : NSObject
@property (nonatomic) BOOL stabilityThresholdEnabled;
@property (nonatomic) int stabilityThreshold;
@end
```

Android

```
public class CaptureExperienceCriteriaHolder {
 public boolean isStabilityThresholdEnabled();
 public void setStabilityThresholdEnabled(boolean stabilityThresholdEnabled);
 public int getStabilityThreshold();
 public void setStabilityThreshold(int stabilityThreshold);
}
```

Page detection mode

When capturing images (not bar codes), you can call the `setPageDetectMode` method to:

- Disable page detection
- Enable page detection only when stability and levelness thresholds are met.
- Enable page detection regardless of thresholds

When one of the second two modes is enabled, page detection events will be fired. The second mode will not allow a user to make any decisions about acceptability in the handler. In "automatic" mode, the control is already in the middle of a taking picture call when the event is raised.

However, when the third mode is enabled, the application can determine if it is OK to take a picture based on the page fully using the frame.

The `PageDetected` event (Android) or delegate (iOS) is raised when a page is detected in the camera preview frame. The detected page coordinates, as well as the preview image used for page detection, are passed to the event handler.

You can use the `setPageDetectMode` method (Android) or the `pageDetect` property (iOS) alone or in combination with the other feedback criteria such as levelness and focus. There are some limitations, see the API Reference guide.

You can also choose to use preview images from the video stream instead of taking discrete images. `useVideoFrame` is a property of the image capture control which determines if the preview image is passed directly to the `imageCaptured` event handler, or if a higher-resolution image is captured first before being passed to those handlers.

Image Capture Frame object

Can be found in UIControls.

The Image Capture Frame object contains properties that define a camera preview frame that serves as a guide for users taking pictures of documents. Much like a picture frame for the document in the viewer window, the guide helps ensure that all page edges are included in the frame. A border is left around

the page edges in the final image, which assists image processing. The application can specify certain characteristics of the frame.

The frame generated by this object can be used to represent the aspect ratio of the target document. For instance, if the document type is for a 5 by 7 page, the frame would have the same form factor, with its longest side parallel to the longest side of the view rectangle.

This gives the user a visual cue of how to frame the picture. The library uses the Image Capture Frame property of the Document type object associated with the Document object passed into Image Capture. The application can also set the color and line attributes of this frame.

The application can also specify a way to make sure the image is taken in a particular landscape or portrait orientation, and a threshold to specify how close to that orientation the camera must be held. If the orientation is within this tolerance, the picture can be taken.

Portrait target frame

There are two ways that the SDK capture experience works with the target frame, depending on the document type:

For generic document types, the SDK supports portrait target frame mode. DocumentCaptureExperience and DocumentDetector are set up for this purpose. This allows the camera preview to be displayed on the device in Portrait mode, and at the same time, the Target Frame to be displayed in a Landscape mode (for example; when capturing Credit Cards while holding the phone in a Portrait orientation).

If the aspect ratio is greater than 1.0, the long edge of the target frame will align with the long edge of the camera preview or image. If the aspect ratio is less than 1.0, the SDK will NOT invert the aspect ratio value. When the aspect ratio is less than 1.0, the long edge of the target frame will align with the short edge of the camera preview or image.

For check and passport documents types, the SDK does not support portrait target frame mode. In other words the SDK does not support an aspect ratio of less than 1.0.

If the aspect ratio is less than 1.0, it will be inverted. Meaning that the long edge of the target frame will always be aligned with the long edge of the camera preview or image.

To set the aspect ratio:

iOS: Set the targetFrameAspectRatio property of kfxKEDCheckDetectionSettings, kfxKEDDocumentDetectionSettings, or KFXPassportDetectionSettings.

Android: Set the setTargetFrameAspectRatio property of CheckDetectionSettings, DocumentDetectionSettings, or PassportDetectionSettings

The following detection settings for the portrait target frame mode are recommended:

- MinFillFraction = 0.65
- MaxFillFraction = 1.1
- ToleranceFraction = 0.07
- MaxSkewAngle = 10

Image Review and Edit control

Can be found in UIControls.

The Image Review and Edit Control object provides a means for the application to display an image (processed or unprocessed) to the user for review or edit. The application explicitly calls the ImageProcessing object to perform the actual image processing operation according to the user's preferences.

The two main features are Zoom and Set Page Boundaries. Zoom allows the user, via platform specific gestures, to magnify the image to see more detail. Set Page Boundaries provides a UI control for setting the boundaries of a selection area (typically the page boundaries) for subsequent cropping and rectangularization by the application.

Note The application is responsible for creating and displaying any screen overlays that offer user feedback, such as "image blurry."

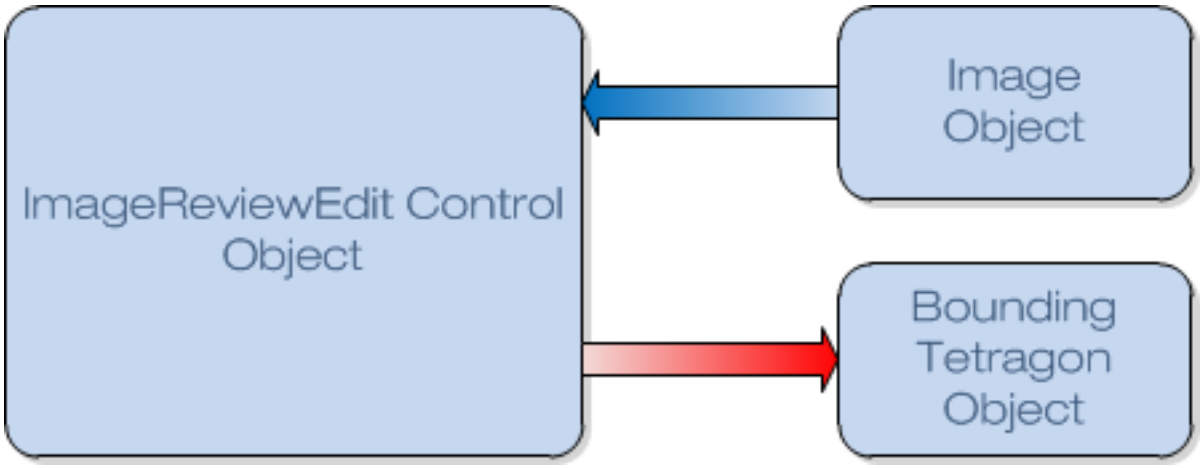


Image Review and Edit Control Object Diagram

The Image Review and Edit Control object shows a view of an image to crop. The library provides this object to view and define cropping points, but it doesn't actually crop the image. To actually crop the image, call the Image Processor.

The Review and Edit object can optionally show a cropping tetragon. The user can move the sides and corners independently to set page boundaries within the image.

Transform an image

In most cases the captured image of a rectangular object will not be rectangular in the image viewer because the camera will not be perfectly parallel to the object being imaged. In some cases the induced perspective may make the image unusable.

In other cases the image may be too large or small.

These issues can be corrected or mitigated by displaying a tetragon shape around the object on the screen using the `cropTetragon` property of the Image Review and Edit control. The application reads out the `cropTetragon` property to retrieve the final corners as adjusted by the user. Then the application supplies the `cropTetragon` via a `BasicSettingsProfile` to the Image Processor, and finally calls the `processImage` to crop and correct the shape of the distorted image.

Highlight extracted data

In order to provide a better user experience, the displayed data can be marked with a highlight in the image preview once the returned coordinates of the extracted data have been used to specify the location and display the highlight in your own application.

The highlights can be invoked by calling `showHighLights (BoundingRect)` after the bitmap or image file path of a processed image has been set in the `ImgReviewEditCntrl` class

Call the `clearHighlights()` method to remove highlights.

Note The highlight feature can only be used on a processed image. It will not work with a raw image.

The `showHighlights` method is invoked from an application to pass an array of `BoundingRect`. This array consists of set of coordinates where the highlights are to be drawn.

The default highlight color of highlight is yellow, and is configurable. Whenever a new color is set, the view is redrawn to reflect that color in the highlights.

Note This feature is not supported on devices with 512 MB of RAM or less. The highlights may not display on such devices.

Indicating the crop area

Indicate the crop area by touching and dragging the edges or corner points of the tetragon.

When the control displays a view of an image, you can adjust the desired crop area by touching a horizontal edge and dragging it vertically to a new position, or by touching a vertical edge and dragging it horizontally up or down to a new position. The library prevents you from dragging an edge beyond the opposite edge. As long as you do not touch and drag the corners, you can adjust the crop area without changing the corner angles.

You can also crop by touching a corner and dragging it to a new position. Corners cannot be dragged across or overlap a side. After dragging corners, the result would be a tetragon, most likely with nonparallel opposite sides.

The result is a set of four point coordinates in pixels. The control stores this data in the Image object as the image is edited. The application uses buttons or some other mechanism to indicate when the user is done manipulating the cropping frame. For instance, if you have a crop button, the button action in the application accesses the point coordinates in the Image object and destroys the view, and then the application decides what to do next.

The application logic can then perform image processing to crop the image. You can decide what the application should do with these four points, as described below.

Crop using a rectangle or tetragon as adjusted by the user

To crop the image using the tetragon as adjusted by the user, the application first reads the `cropTetragon` property to retrieve the final corners as adjusted by the user. Then the application supplies the `cropTetragon` via a `BasicSettingsProfile` to the Image Processor, and finally calls the `processImage` method passing the distorted image. If the tetragon coordinates correspond close enough to a true rectangle (all corner angles near 90 degrees), the image processor does not stretch and fit (rectangularize) the image.

Image object

Can be found in Engines.

The Image object contains an in-memory (and optional) image file-based representation of an image. The image file-based representation is subdivided into stored-file and memory-buffered file representations. Refer to the imageWriteToFileBuffer method of the Image object for more details. The Image object can contain results of operations performed on the Image object by the Image Processor or Image Classifier objects.

- imageQuickAnalysisFeedback contains the results of a quick analysis performed on an input image by the image processor. This is null until quick analysis is performed.
- The classificationResult array contains the results of a classification process performed by the image processor, one result for each possible classification. This array is empty until a classification is performed on an image.
- The imageBarCodes array is empty unless the object is created by the bar code UI control process.

A typical way for an Image object to be created is via the Image Capture Control object. This control will capture the image and return the Image object with an event when all the enabled guidance criteria are met, such as the camera is still enough, level enough and oriented correctly. The Image returned by the Image Capture control is an unprocessed image. After getting the image taken event, the application can perform various editing operations on the image and it can associate that image with any page of any document.

The application should use the `processImage` (ImageProcessor object) and file I/O methods (Image object) carefully. You can do a file write in parallel to an image processing operation, but you cannot do more than one image processing operation in parallel, and attempts to do this will result in a error.

The Image object indicates if the image is represented in a bitmap or in a file (stored or buffered), or both bitmap and file. The application could write the image to a file and also keep the image in memory, but such an approach may encroach on memory limits.

Recommended mime types

- For black and white image storage, the recommended mime type is TIFF. For black and white TIFFs, the Mobile SDK uses G4 compression, which is lossless and much more space efficient than PNG at this color depth.
- For grayscale and color image storage, the recommended mime type is JPEG, unless lossless compression is required by the customer’s application, in which case the recommendation is PNG.

Note Apps should not use the Mobile SDK to create color TIFF images. Attempting to open such images may cause problems on certain mobile OS platforms.

Date and time stamps

The `Image.imageWriteToFile` and `Image.imageWriteToFileBuffer` methods store three EXIF (Exchangeable Image File) tags for date and time into the output image file metadata (JPEG and TIFF only).

- DateTime (code 0x132)
- DateTimeDigitized (code 0x9004)
- DateTimeOriginal (code 0x9003)

If the `imageReadFromFile` method has been used with this Image object, `ImageWriteToFile` writes any JPEG or TIFF date and time stamps from the read image file "as is" into the output file metadata.

Otherwise, `ImageWriteToFile` writes internally generated EXIF metadata into the new JPEG or TIFF file as follows (all times adjusted to GMT):

The EXIF tags DateTime and DateTimeDigitized are generated based on the date and time the image object was created.

EXIF tag DateTimeOriginal is normally generated the same way as the other two EXIF tags, but there is a difference when the given image was created as a result of image processing. In such cases, `imageWriteToFile` will generate the DateTimeOriginal value based on the date/time of the image that was sent to image processing. The input image date/time is normally the date/time that image object was created, but could also be derived from JPEG or TIFF image file metadata used to create that image object via `imageReadFromFile`.

The date/time string is formatted as yyyy:MM:dd HH:mm:ss, as required by the EXIF standard. This format does not accommodate time zones, so the SDK uses the UTC time that the Image object was created, with the time shifted to compensate for the time zone offset.

However, when the `imageReadFromFile()` method has been used with the Image object, the JPEG or TIFF date/time stamp from the original image file metadata is carried as is into the output compressed data stream.

Note Date/time metadata output is supported only with the default Image Processor File I/O Engine.

Memory management

In general, mobile devices have significant memory constraints, and require careful memory management.

This is particularly true with regard to images. Bitmaps, especially those taken with higher pixel count cameras, consume large blocks of memory. When finished with an Image object, the calling application must be sure to call the `imageClearBitmap` and `imageClearFileBuffer` methods on the Image to avoid out-of-memory problems.

Memory management methods may vary by platform, so your application must carefully consider constraints imposed by the execution environment. For example, in Android 2.3 and later, apps may be limited to as little as 16MB of memory on certain devices. Images from some cameras may significantly

exceed this limit. For example, a picture taken with a five megapixel camera may require nineteen megabytes when stored in memory.

Some Android-specific recommendations

Options provided by the Mobile SDK for managing large bitmaps:

- Use the `imageReadFromFile`, `imageReadFromFileBuffer` methods, which take a bitmap scaling parameter, to scale the bitmap when read into memory. The bitmap scaling factor must be greater than 0.1 and less than 1. For example, to scale the image by half, set a value of 0.5

  Note when reading PNG files, the resulting bitmap will be downscaled if it does not fit in memory. An aggressive scaling progression is used, starting with an 0.5 scaling factor on an out-of-memory condition, and halving it successively until the bitmap fits in memory.

- When bitmap scaling is not an option, an alternative is to specify a larger heap size in the application Manifest using the `\a android:largeHeap` attribute as shown below:

```
<application
       android:allowBackup="true"
       android:largeHeap="true"
       android:icon="@drawable/ic_launcher"
       android:name="com.kofax.kmc.kut.utilities.AppContextProvider"
       android:label="@string/application_name"
       android:theme="@style/AppTheme" >
       <activity
           android:name="com.example.enginestest.MainActivity"
           android:label="@string/application_name" >
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />

               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>
   </application>
```

  Apps should use this option only as a last resort, since enabling large heap size does not guarantee a fixed increase in available memory, because some devices are constrained by their total available memory. It's preferable to focus on reducing overall memory usage for improved performance.

- Add the following items to the Android manifest file:

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

Image Processor object

Can be found in Engines.

The ImageProcessor object can process a single image according to settings contained in an `ImagePerfectionProfile` object or a `BasicSettingsProfile` object, but not both simultaneously. The settings in an `ImagePerfectionProfile` object take precedence over those in a `BasicSettingsProfile` object. The output of the ImageProcessor object is a new instance of an Image object that contains the processed image.

An application can use the ImageProcessor object to clean up and perfect images by the application. The `ImageProcessor`'s `processImage` method operates asynchronously. It takes an unprocessed input image and outputs a new image. The image processor does not associate that image with a particular document or page - it is up to the application to do that if it needs to.

Another use of this processor is in concert with a server, such as the Kofax Front Office Server. In this case the KFS administrator creates document types and the associated `ImagePerfectionProfile`. The application uses this profile to process images, create a document and submit it to KFS.

Because the purpose of the image processor is to allow background image processing, there may be no need for image processing progress feedback. However, some large images may take several seconds for the engine to complete processing, so in such cases the application may need to provide some sort of progress feedback. The application can use the processing complete event to detect when to start processing another image.

The `ImageProcessor imageOutEvent` indicates that image processing is complete and provides a reference to the new Image object and a status or error code.

Image Processor Configuration

The Image Processor Configuration (introduced in SDK 3.3) contains a list of properties used to manage the processing settings. Improved cropping algorithms were introduced through a new interface in the Image Processor.

A constructor can be used to create the Image Processor Configuration class from the Image Perfection Profile operation string. This will allow a smooth transition to the new API. Remaining parameters that were not handled and wrapped up with properties are put in the `ippString` variable to be used for legacy processing operations.

The Image Perfection Profile and Basic Settings Profile were deprecated. Image Processor Configuration is a new class which we recommend for use in all image processing cases. Calls to `ImageProcessor.processImage(ImageProcessorConfiguration)` will automatically handle all of the provided image processing operations, both old and new.

Configuration properties:

rotateType - Rotate automatically or in 90 degree increments.

- ROTATE_NONE
- ROTATE_90
- ROTATE_180
- ROTATE_270
- ROTATE_AUTO

outputColorDepth - Use this property to set the desired output color depth.

- BITONAL
- GRAYSCALE
- COLOR

cropType - Crop image to a user specified option.

- CROP_NONE
- CROP_AUTO

targetFrameCropType - Use this property to enable or disable target frame cropping.

- TARGET_FRAME_CROP_OFF
- TARGET_FRAME_CROP_ON

deskewType - Deskew an output image.

- DESKEW_NONE
- DESKEW_BY_DOCUMENT_EDGES
- DESKEW_BY_DOCUMENT_CONTENT

documentDimensions - Use this property to set the length of the shot and longer edges of the original.

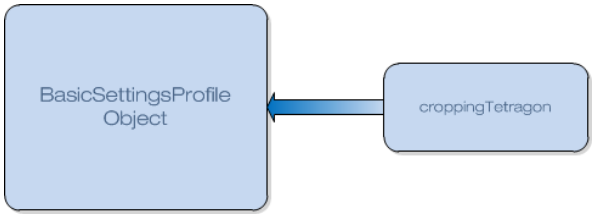
outputDPI - Use this property to set the desired output image DPI (Dots Per Inch). If it is set to 0, then the library will automatically detect the output image DPI and indicate it in the output object.

*ippStrin*g - The IPP string that is used for a secondary processing operation.

BasicSettingsProfile

Note The `BasicSettingsProfile` is deprecated.

The `BasicSettingsProfile` object contains basic settings used to perform image processing, including cropping. These are general purpose settings, unlike the settings in an `ImagePerfectionProfile`, which can be very complicated and targeted to a specific image type. If the application specifies an `ImagePerfectionProfile` to the image processor, any `BasicSettingsProfile` is ignored.



BasicSettingsProfile Object Diagram

ImagePerfectionProfile

Note The `ImagePerfectionProfile` is deprecated.

The `ImagePerfectionProfile` encapsulates advanced image processing settings. These settings may have originated from a KFS Server, a TotalAgility Server or the application itself. The `ImagePerfectionProfile` allows for two ways of specifying settings: the `ipOperations` property, or the `ipOperationsFilePath` property.

If the `ipOperations` string is a non-empty string, the `ImageProcessor processImage` method simply passes the `ipOperations` string as is to the library. In this case, the `ipOperationsFilePath` property is not used in any way.

If the `ipOperationsFilePath` is a non-empty string and the `ipOperations` is an empty string, then the `ImageProcessor processImage` method will load operations string(s) from the specified file. If the specified file contains other contents besides operations string(s), those other contents will be ignored.

It is an error case if both the `ipOperations` and `ipOperationsFilePath` strings are empty strings.

The application can use the `ImagePerfectionProfile`'s properties to enable the processing functions using the keywords described below.

As an example, if the application wants to crop, deskew, binarize, and perform auto-rotation, the application could construct the necessary `ipOperations` string by using the following example code (given in C):

```
strcat(operations, "_DoCropCorrection_");
strcat(operations, "_DoSkewCorrectionPage_");
strcat(operations, "_DoBinarization_");
strcat(operations, "_Do90DegreeRotation_4");
```

As shown in the above example, the `ipOperations` string can contain multiple keywords concatenated together.

Refer to the sample code projects included in this SDK for more examples of code used to prepare the `ipOperations` string.

The following section contains keywords for the `ipOperations` field.

Image operations specification

Most of the image processing modules are disabled by default, but appropriate keywords included in the operations string enable the corresponding modules. In the operations string a leading “_” character indicates the start of a new keyword or parameter. The trailing “_” character may be followed by a setting value: a string in <> or a numeric value. The image processor uses this string to indicate what type of image processing to perform on the input image.

The operations string can explicitly include any number of module activation keywords.

As an example, if a user wants to crop, deskew, binarize and perform auto-orientation, then the corresponding substring would begin with:

```
“_DoCropCorrection__DoSkewCorrectionPage__DoBinarization__Do90DegreeRotation_4”
```

The number 4 after `_Do90DegreeRotation_` indicates automatic orientation/rotation.

The following sections cover keywords for the image processing operation string.

Abort if Page Reject

Operation String Keyword `_DoAbortIfPageReject_`

Header File Macro #define: `DO_ABORT_IF_PAGE_REJECT`

Instructs page detection to return the corresponding error code in case of failure.

Background smoothing

Operation String Keyword `_DoBackgroundSmoothing_`

Header File Macro #define: `DO_BACKGROUND_SMOOTHING`

Performs smoothing of page background for color and grayscale images. Saturates output image to remove artifacts in the background.

Binarization

Operation String Keyword `_DoBinarization_`

Header File Macro #define: `DO_BINARIZATION`

Tells the image processor to output a black and white image.

In order to improve the quality of binary images for documents with low original resolution it is possible to replace `DO_BINARIZATION` with `DO_ENHANCED_BINARIZATION`. This functionality is useful for creation of binary images with resolutions higher that 300 DPI and it results in a binary processed image with the requested high resolution (and correspondingly smoother character contours), but with lower background noise levels.

In order to improve the quality of binary images with very high (300 DPI and up) resolutions and complex backgrounds (such as driver licenses captured with an 8MP camera), you can use a combination of scaling to DPI and enhanced binarization to suppress the background noise. For example, adding `_DoScaleBWImageToDPI_400_DoEnhancedBinarization_` to the `operations_str` will create a 400 DPI binary image with a cleaner background than if using regular binarization.

Note There are several essential parameters controlling binarization, namely:

Property name (case is important)	Description and Parameter range
intelligent_contrast_enabled	Boolean with the default 1 (`TRUE`). If set to 0, will cancel Auto-Contrast.
CBinarize.Contrast_Slider_Pos.Int	Integer within [1,5] with the default = 2 . Controls the aggressiveness of Auto-Contrast determination.
intelligent_brightness_enabled	Boolean with the default=1 (`TRUE`). If set to 0, will cancel Auto-Brightness.
CBinarize.Do_Adv_Clarity.Bool	Boolean with the default=1 (`TRUE`). If set to 0, will cancel Intelligent Cleanup.
CBinarize. Cleanup_Slider_Pos.Int	Integer within [1,5] with the default =2. Controls the aggressiveness of Intelligent Cleanup.
Advanced_Threshold_Dot_Matrix_Image_Enable	Boolean with the default=1 (`TRUE`). For most contemporary documents without any dot-matrix printing, this should be set to 0 to save time spent on this analysis.
global_threshold	Integer within [0,255] with the default 128. Used if Auto-Brightness is off. Sets the brightness threshold for binarization of flat (non-edge) pixels: those with gray levels below 255 - global_threshold will become black, otherwise – white.
edge_threshold	Integer within [0,255] with the default 128. Used if Auto-Contrast is off. Sets the contrast threshold for edge strength to be used by the Dynamic Thresholding. Smaller values will create more edges.

Property name (case is important)	Description and Parameter range
CBinarize.Edge_Aggr.Int	Integer within [0,255] with the default 168. Used by the Dynamic Thresholding.
Threshold.alpha_strength	Integer within [0,255] with the default 128. Used if both Auto-Contrast and Auto-Brightness as well as other advanced features of binarization such as color analysis are off in order to mimic VRS4.5 binarization. Replaces the value of `CBinarize.Edge_Aggr.Int` in the Dynamic Thresholding.
VRS.Analyze.Color.Enable	Boolean with the default=0 (`FALSE`). If set to 1, will enable independent binarization of BLUE, GREEN, and RED channels of a color image with their combination resulting in the final binary image. Takes about 3 times more time than binarization from a grayscale image.
CBinarize.Wei_Blue_To_Gray.Int CBinarize.Wei_Green_To_Gray.Int CBinarize.Wei_Red_To_Gray.Int	If color analysis is off these 3 integer values determine the relative weights of color channels in the conversion of color to gray. The defaults for BLUE, RED, and GREEN are 4, 7 and 5 respectively, meaning that the gray value will be calculated as GRAY=(4*BLUE +7*GREEN+5*RED)/(4+7+5). These values can be used to strengthen or suppress particular colors in the image - for example, in order to make green text stronger `CBinarize.Wei_Green_To_Gray.Int` can be set to 0, or in order to suppress red in the background `CBinarize.Wei_Red_To_Gray.Int` can be set to 1 with the other 2 weights set to 0.

Blank page detection

Operation String Keyword `_DoBlankPageDetection_`

Header File Macro #define: `DO_BLANK_PAGE_DETECTION`

Detects if the page is blank so that it could be deleted by the application.

The following example demonstrates setting the page detection sensitivity.

```
Property name:
VRS.Blank.Page.Content.Sensitivity

Description:
 Integer within [0,255] with the default 128. Controls the sensitivity
 of blank page detection. Higher values are more likely to determine
 the page as non-blank.

Example syntax for adding to operation string:
_LoadInlineSetting_[VRS.Blank.Page.Content.Sensitivity=100]

Output metadata:
 When an image is detected as blank, there will be a "Blank Page" item in
 the ImageMetaData property string of the processed Image as shown in the
 following example. Lack of a "Blank Page" item should be taken as a
 non-blank determination.

{
"Front Side":
       {
```

```
       "Input Image Attributes":
              {
              "Width":2448,
              "Height":3264,
              "Channels":3,
              "BitDepth":24,
              "xDPI":72,
              "yDPI":72
              },
       "Output Image Attributes":
              {
              "Width":2448,
              "Height":3264,
              "Channels":3,
              "BitDepth":24,
              "xDPI":276,
              "yDPI":276
              },
       "Page Detection":
              {
              "Tetragon":
                     {
                     "Corners":
                       {
                       "TLx":   308.1299, "TLy":    86.5922, "TRx":  2308.8644,
                       "TRy":   155.3332, "BLx":   250.8317, "BLy":  3045.1201,
                       "BRx":  2223.6763, "BRy":  3080.1947
                           },
                     "Max Deviation from 90 in degrees": 0.859,
                     "Rectangularized": true
                     }
              },
       "Blank Page":
              {
              "Page is Blank": true
              }
       }
}
```

Blur and illumination check

Operation String Keyword `_DoBlurAndIlluminationCheck_`

Header File Macro #define: `DO_BLUR_AND_ILLUMINATION_CHECK`

Flags blurred, oversaturated and grainy images. Included in Preview mode for camera image capture.

Color detection

Operation String Keyword `_DoColorDetection_`

Header File Macro #define: `DO_COLOR_DETECTION`

Limits binarization to only those pages that do not contain any color. Performs automatic color detection of the input to automatically provide either color or black and white output. The image processor returns a color image if color is detected, or a black and white image otherwise.

Color dropout

Operation String Keyword `_DoColorDropOut_`

Triggers the analysis of the document determining the color of the main background of the page and also the most numerous of the foreground colors. This is assumed to be the color of the form that should be dropped out. The sensitivity can be adjusted.

Property name (case is important)	Description and Parameter range
CColrDrp.Slider_Pos.Int	Integer within [0,255] with default 128. Larger values will increase the sensitivity and result in more expansive dropout.

Crop correction

Operation String Keyword `_DoCropCorrection_`

Header File Macro #define: `DO_CROP_CORRECTION`

Crops to the actual size of the document in the image. Can be used with or without skew correction.

Auto crop

Operation String Keyword `_DoDocumentDetectorBasedCrop__DoCropCorrection_`

Header File Macro #define: `_DODOCUMENTDETECTORBASEDCROP__DOCROPCORRECTION_`

To use the autocrop feature, the above token must be added to the image processing settings of an ImagePerfectionProfile. For additional info on how to create\edit mage processing settings please see ImagePerfectionProfile.

Despeckle

Operation String Keyword `_DoDespeck_n`

Header File Macro #define: `DO_DESPECK`

Removes speckles on the final binary image. n indicates the maximum speck size. Applicable to binary images only.

n=4 is a typical despeck size. The valid range is 1 to 50, with the higher settings removing larger speckles or actual content in the output image.

DeviceType 0

Operation String Keyword `_DeviceType_0`

Specifies non-phone camera specific image processing. Use this when processing an image that has already been processed using phone-camera specific image processing and therefore does not need rectangularization correction.

DeviceType 2

Operation String Keyword `_DeviceType_2`

Specifies phone-camera specific image processing. Note this is included by default.

DocDimLarge

Operation String Keyword `_DocDimLarge_n.nn`

Header File Macro #define: `DOC_DIM_LARGE`

Use this setting to communicate the dimensions (in inches) of the long edge of the document to the image processor.

This is intended to be used only when the document long edge length is explicitly known beforehand. If so, this keyword helps the image processor to more accurately estimate the DPI of the document.

DocDimSmall

Operation String Keyword `_DocDimSmall_n.nn`

Header File Macro #define: `DOC_DIM_SMALL`

Use this setting to communicate the dimensions (in inches) of the short edge of the document to the image processor.

This is intended to be used only when the document short edge length is explicitly known beforehand. If so, this keyword helps the image processor to more accurately estimate the DPI of the document.

Edge cleanup

Operation String Keyword `_DoEdgeCleanup_`

Header File Macro #define: `DO_EDGE_CLEANUP`

Cuts a small frame around the final image in order to clean the fringes of the page. By default edge cleanup is enabled, and this frame is 8 pixels wide regardless of image resolution. The frame width can be adjusted and prorated for DPI.

Property name (case is important)	Description and Parameter range
EdgeCleanup.enable	Boolean with the default=1 (TRUE). Set this to 0 (FALSE) to disable edge cleanup.
CBrdCrop.Crop_Dist.Int	Integer specifying frame width. The default value is 8 pixels.
CBrdCrop.Crop_Dist_Prorate_for_DPI.BOOL	Boolean with the default=0 (FALSE). If set to 1 (TRUE) edge cleanup will prorate CBrdCrop.Crop_Dist.Int by the ratio of image DPI to 200.

Note You must specify both _DoSkewCorrectionPage_ and _DoCropCorrection_ in order for edge cleanup to have any effect.

Note If edge cleanup is performed, the resulting processed image becomes a little smaller while its resolution stays unchanged. In cases when the image dimensions are known, the final image will come out with height and/or width just under the specified value(s). A possible way to keep this discrepancy small is to set CBrdCrop.Crop_Dist_Prorate_for_DPI.BOOL to 1 (TRUE), in which case the default 8 pixels will be equivalent to 8/200 = 0.04".

Enhanced binarization

Operation String Keyword `_DoEnhancedBinarization_`

Header File Macro #define: `DO_ENHANCED_BINARIZATION`

Converts the input image to black and white using an enhanced method to provide better results than those usually seen for standard binarization. This functionality is useful for creation of binary images with resolutions higher than 300 DPI, and it results in a bitonal processed image with the requested high resolution (and correspondingly smoother character contours), but with lower background noise levels.

Note See comments for `DO_BINARIZATION` above.

Final image DPI

Operation String Keyword `_FinalImageDPI_n`

Header File Macro #define: `FINAL_IMAGE_DPI`

Specifies DPI n of the final image. See FINAL_IMAGE SCALING and RESOLUTION for details.

Final image larger pixel dimension

Operation String Keyword `_FinalImageLargerPixelDim_n`

Header File Macro #define: `FINAL_IMAGE_LARGER_PIXEL_DIM`

Specifies pixel size n of the larger side of the final image. See FINAL_IMAGE SCALING and RESOLUTION for details.

Final image smaller pixel dimension

Operation String Keyword `_FinalImageSmallerPixelDim_n`

Header File Macro #define: `FINAL_IMAGE_SMALLER_PIXEL_DIM`

Specifies pixel size n of the smaller side of the final image. See FINAL_IMAGE SCALING and RESOLUTION for details.

Find graphic lines

Operation String Keyword `_DoFindGraphicLines_`

Header File Macro #define: `DO_FIND_GRAPHIC_LINES`

Finds horizontal and vertical graphic lines in the image.

Find text hand print

Operation String Keyword `_DoFindTextHP_`

Header File Macro #define: `DO_FIND_TEXT_HP`

Finds lines of handwriting and returns their coordinates. Can be used for signature detection on checks. It can be combined with `_ProcessCheckFront_`. For successful hand print detection, the application must ensure that the image is deskewed and cropped. This is because noisy areas in the perimeter of the image can easily be mistaken for hand printing. The application must include the tokens `_DoSkewCorrectionPage_` and `_DoCropCorrection_` to perform deskew and crop unless it is known that these operations have already been performed on the image. See also MICR Recognition and Hand Print Detection.

Since the purpose of signature detection in check images is only to flag those of them without a signature and not verification of its validity, any cursive writing detected in the bottom right area of the image qualifies as signature. In order to obtain the metadata allowing the calling application to do that, the `operations_str` used in the image perfection profile should include `_DoFindTextHP_`.

Related properties:

```
CDetectMpHp.RegionOfInterestPercX1.Int
CDetectMpHp.RegionOfInterestPercX2.Int
CDetectMpHp.RegionOfInterestPercY1.Int
CDetectMpHp.RegionOfInterestPercY2.Int
```

Description:

These settings define the region of the detected page in which the hand print finding is to be performed. The point (X1, Y1) defines the top left point of the rectangular region, and (X2, Y2) defines the bottom right point of that region. The X1 and X2 values are expressed as a percentage of the width of the detected page. The Y1 and Y2 values are expressed as a percentage of the height of the detected page. The defaults are X1=0, Y1=0, X2=100, Y2=100, which specifies the entire detected page is to be searched. As an example, to specify the bottom right quarter of the detected page, specify X1=50, Y1=50, X2=100, Y2=100.

Depending on the image, a large amount of metadata may be generated if the entire detected page is searched for hand printing. Not only is it laborious for the app to search through all this metadata, it may in some cases overflow the buffer allocated by the SDK to contain the metadata. For these reasons it is highly recommended to use the above RegionOfInterest settings to confine the hand print searching to the relevant area of the detected page.

In the following example, the text line of “MI” type reflects the results of MICR recognition, and the text line of “HP” type corresponds to the signature.

```
"Text Lines":
  {
  "Num": 2,
  "Lines":
    [
   { "Index":    0, "Type": "MI", "TLx":  112, "TLy":  709, "TRx": 1223, "TRy":
      709, "BLx":  112, "BLy":  753, "BRx": 1223, "BRy":  753, "Label":
      "MICR", "OCR_System": "MICR", "OCR_Template": "", "OCR Length":   27, "OCR
 Data":
      "C222371863C 0251508 30P   3587",
     "Num Words": 1,
     "Words":
    [
   {"TLx":  112, "TLy":  709, "TRx": 1223, "TRy":  709, "BLx":  112, "BLy":  753,
 "BRx":
      1223, "BRy":  753, "Length":   27, "Word": "C222371863C 0251508 30P   3587" }
    ]
   },
   { "Index":    1, "Type": "HP", "TLx":  955, "TLy":  508, "TRx": 1555, "TRy":
      508, "BLx":  955, "BLy":  685, "BRx": 1555, "BRy":  685, "Label": "",
"OCR_System":
      "Preprocess", "OCR_Template": "", "OCR Length":    0, "OCR Data": "",
     "Num Words": 0,
     "Words":
    [
    ]
   }
   ]
  }
```

Find text machine print

Operation String Keyword `_DoFindTextMP_`

Header File Macro #define: `DO_FIND_TEXT_MP`

After binarization looks for approximately horizontal lines of text and returns their coordinates.

Gray output

Operation String Keyword `_DoGrayOutput_`

Header File Macro #define: `DO_GRAY_OUTPUT`

Tells the image processor to output a grayscale image. The request to convert the output image to grayscale is ignored when the incoming image is binary.

Hole fill

Operation String Keyword `_DoHoleFill_`

Header File Macro #define: `DO_HOLE_FILL`

Fills punch holes on document edges in order to match the surrounding page background. Performed for dark backgrounds only.

Illumination correction

Operation String Keyword `_DoIlluminationCorrection_`

Header File Macro #define: `DO_ILLUMINATION_CORRECTION`

Attempts to equalize the brightness and hue of page background. The boolean flag set to `TRUE` by this keyword can be alternatively set by specifying the value of the parameter `CSkewDetect.correct_illumination.Bool`. By default it is set to `TRUE` for Mobile; however, if it is known that the documents have no uniform background it is better to save processing time and avoid problematic corrections by setting this flag to `0` (`FALSE`).

Load operations string tokens from file

Operation String Keyword `_LoadOperationsStringTokensFromFile_<filename>`

Tells the image processor to load operations tokens from a file. The image processor opens the specified file and looks for a section in it like the following example:

```
<OPERATIONS_STRING_TOKENS>
<Property Name="OPERATIONS_STRING_TOKENS" Value="_DoSkewCorrectionPage_" />
<Property Name="OPERATIONS_STRING_TOKENS" Value="_DoCropCorrection_" />
<Property Name="OPERATIONS_STRING_TOKENS" Value="_DoEnhancedBinarization_" />
</OPERATIONS_STRING_TOKENS>
```

LoadSetting

Operation String Keyword `_LoadSetting_<filename>`

Use to load non-default processing parameters from a file.

For example, `_LoadSetting_</mnt/sdcard/non_defaults.xml>` will load the contents of a `non_defaults.xml` from an SD card of an Android device.

In the specified file, the image processor looks for the keyword `<VINSET>`, which indicates the beginning of the section to load. Each subsequent line of xml is loaded until `</VINSET>`, is found.

The calling application can indicate that it is also necessary to read lines between `<VINSET_XXX>` and `</VINSET_XXX>`, where the XXX string can be passed through the ipOperations string by adding `_ProcessID_` followed by the string. For example, if you use `_LoadSetting_<non_defaults.xml>_ProcessID_<DL_AL_T1F>`, all lines of `non_defaults.xml` between `<VINSET_DL_AL_T1F>` and `</VINSET_DL_AL_T1F>` will be also loaded.

Note Refer to the most recent version of the Kofax Mobile Demo sample application source code for an example of a properly formatted XML file.

Load settings from file operation string keyword

Operation String Keyword `LoadSettingsFromFile_<filename>`

Use to load non-default processing parameters from a file. The filename extension can be .ini, .xml. or .json, with lines appropriately formatted, to load correctly.

Load inline setting

Operation String Keyword `_LoadInlineSetting_`

Use to load a non-default processing parameter from character string following the keyword. The character string can be in either .ini, .xml. or .json format specified respectively by [ ], < >, or { } characters within which it must be enclosed.

INI-style example (preferred):

```
_LoadInlineSetting_[intelligent_contrast_enabled=0]
```

JSON-style example:

```
_LoadInlineSetting_{"intelligent_contrast_enabled":0}
```

XML-style example:

```
_LoadInlineSetting_<Property Name="intelligent_contrast_enabled" Value="0"/>
```

No page detection

Operation String Keyword `_DoNoPageDetection_`

Header File Macro #define: `DO_NO_PAGE_DETECTION`

Informs the image processor that the incoming image has been already deskewed and cropped, so no page detection is necessary.

Image processing always begins with page detection. Even if deskew to page or deskew to content, as well as crop, are not requested, page detection is still performed unless it is cancelled by setting `_DoNoPageDetection_`. The reason for that is the need to know where the page is in order to detect content like punch holes within it, to decide whether the page is blank, or to analyze the image only inside the page while determining how better to binarize it.

Note The important difference between the `DO_NO_PAGE_DETECTION` and the absence of the `DO_SKEW_CORRECTION_PAGE`, `DO_SKEW_CORRECTION_ALT`, and `DO_CROP_CORRECTION` controls is that their absence does not cancel page detection, just skew and crop correction. Normally page detection is always performed because it is necessary for other operations.

Process check front

Operation String Keyword `_ProcessCheckFront_`

Header File Macro #define: `PROCESS_CHECK_FRONT`

Enables the page detection algorithm specific to bank check fronts. Enables MICR recognition and reporting of MICR line in the metadata. It can be combined with `_DoFindTextHP_`. See also MICR Recognition and hand print Detection.

Note that no error is generated for lack of MICR on a check front.

Process check back
Operation String Keyword `_ProcessCheckBack_`
Header File Macro #define: `PROCESS_CHECK_BACK`
Enables MICR recognition and reporting of MICR line in the metadata. It can be combined with `_DoFindTextHP_`. See also MICR Recognition and Hand Print Detection.
If MICR is found on the check back, MICR is reported in the metadata and no error is generated.

Process ID
Operation String Keyword `_ProcessID_`
Header File Macro #define: `PROCESS_ID`
This is used with `_LoadSettingsFromFile_`. See that section for a description.

Recognize text MICR
Operation String Keyword `_DoRecognizeTextMICR_`
Header File Macro #define: `DO_RECOGNIZE_TEXT_MICR`
This is a non-preferred synonym for the keyword `_ProcessCheckFront_`.

Remove graphic lines
Operation String Keyword `_DoRemoveGraphicLines_`
Header File Macro #define: `DO_REMOVE_GRAPHIC_LINES`
Finds and removes graphic lines from the processed binary image.

Rotate AUTO
Operation String Keyword `_Do90DegreeRotation_4`
Header File Macro #define: `DO_ROTATE_AUTO`
Automatically rotate the image so the text is oriented normally.

Rotate none
Operation String Keyword `_Do90DegreeRotation_0`
Header File Macro #define: `DO_ROTATE_NONE`
Do not rotate the image.

Rotate 90
Operation String Keyword `_Do90DegreeRotation_3`
Header File Macro #define: `DO_ROTATE_90`
Rotate the image 90 degrees clockwise.

Rotate Auto + 90
Operation String Keyword `_Do90DegreeRotation_7`

Automatically rotate the image so the text is oriented normally and then rotate the image an additional 90 degrees clockwise.

Rotate Auto + force landscape 90

Operation String Keyword `_Do90DegreeRotation_9`

Automatically rotate the image so the text is oriented normally and then, if necessary to make the output image be landscape orientation, rotate the image an additional 90 degrees clockwise.

Rotate 180

Operation String Keyword `_Do90DegreeRotation_2`

Header File Macro #define: `DO_ROTATE_180`

Rotate the image 180 degrees.

Rotate Auto + 180

Operation String Keyword `_Do90DegreeRotation_6`

Automatically rotate the image so the text is oriented normally and then rotate the image another 180 degrees.

Rotate 270

Operation String Keyword `_Do90DegreeRotation_1`

Header File Macro #define: `DO_ROTATE_270`

Rotate the image 270 degrees clockwise.

Rotate Auto + 270

Operation String Keyword `_Do90DegreeRotation_5`

Automatically rotate the image so the text is oriented normally and then rotate the image another 270 degrees clockwise.

Rotate Auto + force landscape 270

Operation String Keyword `_Do90DegreeRotation_8`

Automatically rotate the image so the text is oriented normally and then, if necessary to make the output image be landscape orientation, rotate the image an additional 270 degrees clockwise.

90 degree rotation

Operation String Keyword `_Do90DegreeRotation_n`

Header File Macro #define: `DO_90_DEGREE_ROTATION`

Performs image rotation in 90 degree increments or automatically, based on image content.

1=270 degrees clockwise; 2=180 degrees; 3=90 degrees; 4=automatic; 5=auto+270 degrees clockwise; 6=auto+180 degrees; 7=auto+90 degrees clockwise; 8=auto+270 degrees clockwise if necessary to force landscape; 9=auto+90 degrees clockwise if necessary to force landscape

Image rotation is performed on the final cropped and deskewed image if these options are included. If automatic rotation (4 through 9) is selected, then the image processor will rotate the image according

to content found in the image, so that the image is right side up, and the additional (auto+) rotation is performed if specified. Rotation is normally used with crop and deskew enabled.

Scale after page detection

Operation String Keyword `_DoScaleAfterPageDetection_n`

Header File Macro #define: `DO_SCALE_AFTER_PAGE_DETECTION`

The original image can be scaled down to reduce time spent on subsequent image processing.

This feature is used to speed up processing of high resolution images of small documents like business cards and driver licenses. The scaling (only down) is always performed after page detection which is using the original image. The integer factor following this keyword is based on fractions of 60, so in order to scale down by 2x its value should be 120; in order to scale down by 3:2 it should be 90; in order to scale down by 4:3 it should become 80. Scaling down by an integer, 3:2, and 4:3 is performed by dedicated routines and is much quicker than the generic scaling.

Scale black/white image to DPI

Operation String Keyword `_DoScaleBWImageToDPI_n`

Header File Macro #define: `DO_SCALE_BW_IMAGE_TO_DPI`

Desired DPI of binary processed image.

See comments for `DO_SCALE_CG_IMAGE_TO_DPI` above.

Scale color/gray image to DPI

Operation String Keyword `_DoScaleCGImageToDPI_n`

Header File Macro #define: `DO_SCALE_CG_IMAGE_TO_DPI`

n=desired DPI of color or gray processed image.

It is possible to request different resolutions of processed images depending on whether they come out in color/gray or binary. This feature is most useful when the user requests binarization based on color detection because in order to match the success of OCR from color or gray it is necessary to use binary images of at least 1.5 times the resolution.

For more information see DPI Estimation.

Scale image to DPI

Operation String Keyword `_DoScaleImageToDPI_n`

Header File Macro #define: `DO_SCALE_IMAGE_TO_DPI`

n=desired DPI of processed image. Suggested values: [150,200,240,300] .

Sharpen

Operation String Keyword `_DoSharpen_n`

Header File Macro #define: `DO_SHARPEN`

Sharpens processed color or gray image, n: 1=a little, 2=more, 3=most.

Skew correction by content

Operation String Keyword `_DoSkewCorrectionAlt_`

Header File Macro #define: `DO_SKEW_CORRECTION_ALT`

Performs skew correction based on content.

Skew correction by page edges

Operation String Keyword `_DoSkewCorrectionPage_`

Header File Macro #define: `DO_SKEW_CORRECTION_PAGE`

Performs skew correction based on edges.

For both types of skew correction, if any of the page corners are outside the image frame, skew correction has to fill the missing areas with some color. By default, skew correction fills these areas with the median color of the surface the document was photographed against. This behavior can be modified and the desired color provided by the following settings:

Property name (case is important)	Description and Parameter range
CSkwCor.Fill_Color_Scanner_Bkg.Bool	Boolean with the default=1 (TRUE). By default, missing areas are filled with the median color of the surface the document was photographed against. If set to 0 (FALSE), skew correction uses the color specified by the following settings.
CSkwCor.Fill_Color_Red.Byte CSkwCor.Fill_Color_Green.Byte CSkwCor.Fill_Color_Blue.Byte	Integers within [0,255]. The default=0 (black) for all of them. Used only if CSkwCor.Fill_Color_Scanner_Bkg.Bool is set to 0 (FALSE). .

Resolution of inconsistencies

If the `operations_str` contains contradictory requests, such as `DO_BINARIZATION` and `DO_GRAY_OUTPUT`, the image processor resolves the conflicts as follows:

- `DO_BINARIZATION` and `DO_ENHANCED_BINARIZATION` take precedence over `DO_GRAY_OUTPUT` regardless of their order in the operations string.
- `DO_SKEW_CORRECTION_PAGE` and `DO_SKEW_CORRECTION_ALT` can coexist. If they are both present, then deskew to content will be done if the found page gets rejected.
- In case the same keyword or parameter name is followed by a different parameter value, for example, `_DoScaleBWImageToDPI_200_DoScaleBWImageToDPI_300`, the last value will be used.
- All individual parameter values loaded from substrings beginning with `LOAD_INLINE_SETTING` or `LOAD_SETTINGS` are loaded in order they are mentioned in the `operations_str`, so if the same parameter name is mentioned again, the last value will be used.

Rectangularization

Rectangularization is an image processing feature that corrects an image taken from a camera that is tilted in reference to the source document. In this case, due to projective effects, the document page may have a tetragon shape with slightly curved sides. In the processed output image, the page is converted to a rectangle, and the content of the page is stretched appropriately.

Rectangularization detection and correction are automatically performed as part of page detection.

For driver licenses only, the edge detection algorithm uses a dynamic threshold parameter to enhance edge detection for low contrast backgrounds. The dynamic threshold is implemented by invoking multiple runs of the edge detection algorithm with different thresholds in order to get the best result.

Image processing: date and time stamps

The SDK also contains a provision to add the date and time stamps to the output file metadata when the image is processed. For more information on the date and time stamps, see Date and time stamps. For specific information about adding these stamps during image processing, see the API Reference help system.

DPI estimation

Images coming from cameras do not have a defined resolution, so when the image processor first receives them their DPI is estimated based on the assumption that the document is letter-size and fits with a small margin within the frame of the image. In reality, the document can have a higher or lower actual resolution depending on its size, so the image processor uses a mixed strategy in order to estimate resolution in absence of any direct information (such as distance to object determined by auto-focus) from the camera.

This information can be provided via the image the perfection profile by including one or both of the keywords, `_DocDimSmall_` and/or `_DocDimLarge_` followed by the corresponding dimensions in inches. For example, the dimensions of a standard US driver license can be set as `_DocDimSmall_2.125_DocDimLarge_3.375`. If only one of them is known, it is still sufficient, but if both are known it helps to correct slight discrepancies caused by a lack of precision in the page detection results.

Alternatively, it is possible to set the value of `CSkewDetect.document_size.Int`. This parameter refers to the larger dimension of the document and can be set to 1 (small), 2 (medium), or 3 (large), and it will be used to set the approximate value of `_DocDimLarge_`.

Finally, if no information regarding document dimensions is available, DPI estimation estimates the resolution of the document based on the found content.

Note If it is known that the incoming document is letter-size and that it fills the camera frame with just small margins, then it is simply wasting processing time and the whole effort can be canceled by setting `CSkewDetect.estimate_dpi.Bool` to 0.

If both dimensions of the document are known, the image processor will try to match the aspect ratio of the found document to that calculated from the dimensions provided by the caller. In order to prevent damage due to errors in page detection, the estimated aspect ratio (average width and height as seen in the photo) is compared to the aspect ratio of the provided dimensions. If the difference is greater than the parameter `CSkewDetect.document_max_aspect_diff.double` the estimated aspect ratio is left as final. The default value of `CSkewDetect.document_max_aspect_diff.double` is 0.25.

The application can see the estimated dpi in the `imageDPI` property of the output Image object referenced in the `imageOutEvent`. The `imageOutEvent` is associated with the `processImage` method of the `ImageProcessor`.

Process progress feedback

To provide progress information, the application needs to implement a process progress event handling method.

The image processor generates events as the percent of completed status updates, so the application can display actual progress. The application event handler could use the percent complete and update a progress bar or keep a spinner spinning or design some other feedback display.

The image processor also generates an event for process complete. When the application processes this event, it can report this event and then access the output image, move it to a Document object, write it to disk or perform other operations with it.

Cancel image processing

The application developer may use the cancel method in the image processor to cancel image processing. If the application includes a button to call the cancel method, the application users can cancel image processing when started by any profile, or to cancel quick analysis.

The ability to cancel image processing allows the application to respond to low-memory events or stop processing when the user decides to capture another image while the first is being processed in the background. To do this, asynchronously call the cancel method in the image processor. Note that there may be a delay between cancelling and the image out event. This happens because the image processor will only cancel the balance of processing tasks on functional boundaries. The image out event will indicate that the processing was cancelled if the image processing operation was incomplete.

Image processing is an asynchronous operation. Also, the application can call the cancel method in the image processor at any time. If the application calls the cancel method when the image processing is nearly complete, the cancel method call may not cancel the operation. Instead, the library operation will finish and report final completion status.

Note The cancel method ignores all calls when no image processing operation is in progress. In this case, the method will not return an error. The cancel condition will not be persisted, so that a subsequent image processing operation will not be immediately cancelled.

Queue management

If the application needs to have input and output Image queues, the application must create and manage these on its own. The application may find it helpful to use the `imageID` and `imageSourceID` properties of the Image object if the application is implementing some type of image queue management.

To help prevent problems, the Image Processor detects and indicates errors for the following conditions:

- `ImageProcessor` is already busy doing a previously requested image processing operation.
- File I/O facility is already busy doing a previously requested File I/O operation.

Note There is no need to wait for a current image processing task to complete before changing the `BasicSettingsProfile` property or the `ImagePerfectionProfile` for the `ImageProcessor`. The current process will complete with the properties and settings in effect when it started. Subsequent changes will take effect the next time the Image Processor is invoked.

Final_Image scaling and resolution

By default, image processing produces an output image with pixel dimensions (image width and height) and DPI as determined by the image processor based on characteristics of the input image and on the specified image processing operations.

The application can specify `_FinalImageSmallerPixelDim_n` or `_FinalImageLargerPixelDim_n` (or both) in the image processing operation string to precisely control one or both of the output image pixel dimensions. This can be useful if, for example, the final image is used as an input to another application with precise requirements for image height or width in pixels. The image processor performs the adjustment by scaling the output image as necessary immediately before returning it to the application.

If the application specifies `_FinalImageDPI_n` in the image processing operation string in addition to one or both final image pixel dimensions, the image processor scales the output image based on the specified pixel dimension(s), and then the image processor sets the output image DPI value to the specified value.

If the application specifies `_FinalImageDPI_n` but does not specify either of the final image pixel dimensions, the image processor scales the output image based on the specified final image DPI, and then the image processor sets the output image DPI value to the specified value.

MICR Recognition and hand print detection

`ImagePerfectionProfile` tokens can be used to activate MICR recognition and hand-printed text detection.

MICR recognition

The following `ImagePerfectionProfile` operations tokens are used to activate MICR recognition:

ProcessCheckFront
This is required in order to find and recognize the MICR line text. It can be combined with `_DoFindTextHP_`.

MICR parsing

The results returned by the real-time check detection class include properties for the routing number, account number, and check number. Specifically, the returned fields are:

1. **AuxiliaryOnUs** When this field is present (usually only on business checks), it usually contains the check number.
2. **EPC** This field is used for "specific purposes" and cannot be used without written authorization from ASC (Accredited Standards Committee) X9B.
3. **RoutingNumber** Officially this is called the transit field, but it is only used for the routing number.

4. **OnUs** Returns all characters from the MICR, from the transitNumber to the amount (if present), or to the right of the string if the amount is not present.
5. **On-Us1**, **On-Us2** The On-Us field are convenience parses provided where, ignoring any left-leading onus symbols in the OnUs field, onus1 is the part to the left of any remaining onus symbol and onus2 is the part to the right.
6. **Amount** If present, a zero padded string of the encoded check amount, in cents.
7. **Account Number** Returns numeric value only, no alpha characters or spaces
8. **CheckNumber** Returns numeric value only, no alpha characters or spaces.
9. **TransitNumber** Returns numeric value only, no alpha characters or spaces.

Hand print detection

The following `ImagePerfectionProfile` operations tokens are used to activate hand print detection:

DoFindTextHP
This is required in order to find hand-printed lines of text. It can be combined with `_ProcessCheckFront_`.
For successful hand print detection, the application must ensure that the image is deskewed and cropped. This is because noisy areas in the perimeter of the image can easily be mistaken for hand printing. The application must include the tokens `_DoSkewCorrectionPage__DoCropCorrection_` to perform deskew and crop unless it is known that these operations have already been performed on the image.

Example metadata results

After image processing is completed for an Image object, the application can access the hand print detection and MICR recognition results in the imageMetaData property of the processed Image.

In general, the ImageProcessor formats the imageMetaData property as a JSON string. The following imageMetaData example shows combined hand print detection and MICR recognition results:

```
{
"Front Side":
 {
 "Output Image Attributes":
  {
  "Width":2953,
  "Height":1356,
  "Channels":3,
  "BitDepth":24,
  "xDPI":318,
  "yDPI":318
  },
 "Text Lines":
  {
  "Num": 4,
  "Lines":
   [
   { "Index":    0, "Type": "MI", "TLx":  139, "TLy": 1185,
                                  "TRx": 1905, "TRy": 1185,
                                  "BLx":  139, "BLy": 1259,
                                  "BRx": 1905, "BRy": 1259,
     "Label": "MICR",
```

```
     "OCR Length": 29, "OCR Data": "C123454321C 0123454321P 09999"},

   { "Index":    1, "Type": "HP", "TLx":  148, "TLy":  820,
                                  "TRx": 1182, "TRy":  820,
                                  "BLx":  148, "BLy": 1254,
                                  "BRx": 1182, "BRy": 1254,
     "Label": "", "OCR Length":    0, "OCR Data": "" },

   { "Index":    2, "Type": "HP", "TLx": 1607, "TLy":  906,
                                  "TRx": 2560, "TRy":  906,
                                  "BLx": 1607, "BLy": 1146,
                                  "BRx": 2560, "BRy": 1146,
     "Label": "", "OCR Length":    0, "OCR Data": "" },

   { "Index":    3, "Type": "HP", "TLx":  160, "TLy":  194,
                                  "TRx": 2842, "TRy":  194,
                                  "BLx":  160, "BLy":  783,
                                  "BRx": 2842, "BRy":  783,
     "Label": "", "OCR Length":    0, "OCR Data": "" }
   ]
  },
 "Auto Orientation":
  {
  "Auto Orientation has been done": true
  }
 }
}
```

With regard to the above sample, note the following:

- The Index field contains the sequential number of the found text line.
- The Type field identifies the type of text line machine print (MP), or hand print (HP). A MICR text line is categorized as type MP. Type HP lines can appear only if hand print detection has been performed.
- The pixel coordinates of the text line bounding box are contained in the subsequent 8 fields, from TLx to BRy.
- The Label field value of MICR identifies the text line as being the MICR line.
- The OCR Length and OCR Data fields contain the number of recognized characters and the corresponding character string. These, and the Label field, are currently used only for MICR recognition.

Note The following special characters can appear in the MICR line:

- Transit (⑆)
- Amount (⑇)
- On-us (⑈)
- Dash (⑉)

In the OCR data field of the metadata, these special characters are represented by normal characters as follows:

- Transit: letter "C"
- Amount: slash "/"
- On-Us: letter "P"
- Dash: hyphen "-"

Code Sample

Following are Android and iOS code snippets for MICR reading and hand print detection.

MICR reading

Android:

```
JSONObject mJSONObject = new JSONObject(this.metadata);
JSONObject mFrontSideObject = mJSONObject.getJSONObject("Front Side");
JSONObject mImageDimObject = mFrontSideObject.getJSONObject("Output Image Attributes");
mProcImageWidth = mImageDimObject.getInt("Width");
mProcImageHeight = mImageDimObject.getInt("Height");
mProcImageDPI = mImageDimObject.getInt("xDPI");
JSONObject mLinesObject = null;
JSONArray mLinesCordinates = null;
try {
    mLinesObject = mFrontSideObject.getJSONObject("Text Lines");
    mLinesCordinates = mLinesObject.getJSONArray("Lines");

    for(int i=0;i< mLinesCordinates.length();i++)
    {
     if("MICR".equals(mLinesCordinates.getJSONObject(i).getString("Label"))&& !
mMICRFound)
        {
            mOCRData = mLinesCordinates.getJSONObject(i).getString("OCR Data");
            mMICRTLy = mLinesCordinates.getJSONObject(i).getInt("TLy");
            mMICRBLy = mLinesCordinates.getJSONObject(i).getInt("BLy");
            mMICRFound = checkMICR(mOCRData);
        }
    }
} catch (JSONException e) {
}

for(int i=0;i< mLinesCordinates.length();i++)
{
 f(("HP").equals(mLinesCordinates.getJSONObject(i).getString("Type")) && !
mSignatureFound)
    {
        mSignatureFound = true;
    }
}

// Reject bad MICR reads (1) by checking height, and (2) by pattern matching the
//  MICR data.
public boolean checkMICR(String OCRData)
{
    final int MIN_MICR_HEIGHT = 8;
    boolean result = false;
    if ((mMICRBLy - mMICRTLy) >= MIN_MICR_HEIGHT) {
        if (OCRData.matches(".*C\\d{9}C.*")) {
            result = true;
        }
    }
    return result;
}
```

iOS:

```
//Function which talks to RTTI to extract amount from a check
```

```
-(void)extractCheckAmount:(NSString*)imagePath { //imagePath contains path of tiff
 image

    if([imagePath isEqualToString:@""] || [imagePath isEqual:nil]){

        return;
    }

 NSString *urlString = @"http://mobiledemo.kofax.com"; //IP address/ host name of
 extraction server

    NSURL *checkExtractionServerURL = [NSURL URLWithString:[NSString stringWithFormat:
    @"%@/mobilesdk/api/CheckDeposit", urlString]]; //Appending check deposit extraction
    service path to the original Extraction Server URL

    NSData *checkData = [NSData dataWithContentsOfFile:imagePath];
    //Storing tiff image as NSData object


 //Forming SOAP request to talk to the server
    NSMutableURLRequest  *urlRequest = [NSMutableURLRequest requestWithURL:
    checkExtractionServerURL cachePolicy:NSURLRequestReloadRevalidatingCacheData
    timeoutInterval:30];
    [urlRequest setHTTPMethod:@"PUT"];
    [urlRequest setValue:@"application/json" forHTTPHeaderField:@"Accept"];
    [urlRequest setValue:@"image/tiff" forHTTPHeaderField:@"Content-Type"];
    [urlRequest setHTTPBody:checkData];


    // Preparing Connection to the server
    NSURLConnection *checkConn = [[NSURLConnection alloc] initWithRequest:urlRequest
    delegate:self startImmediately:NO];


    [checkConn scheduleInRunLoop:[NSRunLoop mainRunLoop] forMode:NSRunLoopCommonModes];
    [checkConn start];  // Starting connection to server


 //We will be getting  the response/Data in the NSURL Delegate methods viz.
      // - (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data
      // -(void)connectionDidFinishLoading:(NSURLConnection *)connection
}



//Function which talks to RTTI to extract details from a remittance coupon

-(void)extractBillDetails:(NSString*)imagePath{ //imagePath contains path of tiff image

 if([imagePath isEqualToString:@""] || [imagePath isEqual:nil]){

        return;
    }

 NSString *urlString = @"http://mobiledemo.kofax.com";
 //IP address/ host name of extraction server

    NSURL *billExtractionServerURL = [NSURL URLWithString:[NSString stringWithFormat:
    @"%@/mobilesdk/api/billpay", urlString]]; //Appending bill pay extraction service
    path to the original Extraction Server URL

    NSData *billData = [NSData dataWithContentsOfFile:imagePath]; //Storing tiff image
    as NSData object
```

```
    //Forming SOAP request to talk to the server
    NSMutableURLRequest  *urlRequest = [NSMutableURLRequest requestWithURL:
    billExtractionServerURL cachePolicy:NSURLRequestReloadRevalidatingCacheData
    timeoutInterval:30];
    [urlRequest setHTTPMethod:@"PUT"];
    [urlRequest setValue:@"application/json" forHTTPHeaderField:@"Accept"];
    [urlRequest setValue:@"image/tiff" forHTTPHeaderField:@"Content-Type"];
    [urlRequest setHTTPBody:billData];

 // Preparing Connection to the server
    NSURLConnection *billConn = [[NSURLConnection alloc] initWithRequest:urlRequest
    delegate:self startImmediately:NO];

 [billConn scheduleInRunLoop:[NSRunLoop mainRunLoop] forMode:NSRunLoopCommonModes];
    [billConn start];  // Starting connection to server


 //We will be getting  the response/Data in the NSURL Delegate methods viz.
      // - (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data
      // -(void)connectionDidFinishLoading:(NSURLConnection *)connection
}
```

Hand print detection

Android

```
// Meta Data we get after processing front of check is as follows

/*
{
"Front Side":
 {
 "Input Image Attributes":
  {
  "Width":2448,
  "Height":3264,
  "Channels":3,
  "BitDepth":24,
  "xDPI":200,
  "yDPI":200
  },
 "Output Image Attributes":
  {
  "Width":2837,
  "Height":1310,
  "Channels":1,
  "BitDepth":1,
  "xDPI":475,
  "yDPI":475
  },
 "Page Detection":
  {
  "Tetragon":
   {
   "Max Deviation from 90 in degrees": 3.425,
   "Rectangularized": true
   }
  },
 "Text Lines":
  {
  "Num": 3,
  "Lines":
   [
   { "Index":    0, "Type": "HP", "TLx": 1828, "TLy":  726, "TRx": 2194, "TRy":  726,
```

```
   "BLx": 1828, "BLy": 1023, "BRx": 2194, "BRy": 1023, "Label": "", "OCR Length":
   0, "OCR Data": "" },
   { "Index":    1, "Type": "HP", "TLx": 2204, "TLy":  502, "TRx": 2318, "TRy":  502,
   "BLx": 2204, "BLy":  698, "BRx": 2318, "BRy":  698, "Label": "", "OCR Length":
   0, "OCR Data": "" },
   { "Index":    2, "Type": "MP", "TLx":  187, "TLy": 1139, "TRx": 2311, "TRy": 1139,
   "BLx":  187, "BLy": 1217, "BRx": 2311, "BRy": 1217, "Label": "MICR", "OCR Length":

   21, "OCR Data": "C000067895C 12345678P" }
   ]
  },
 "Auto Orientation":
  {
  "Auto Orientation has been done": true
  }
 }
}

*/



Check for Signature Front and Back (DocID 0 == Front of Check, DocID 1 == Back of
 Check)
Index values received as json parameters are sent to the bellow function.

Ex:
{ "Index":    0, "Type": "HP", "TLx": 1828, "TLy":  726, "TRx": 2194, "TRy":  726,
 "BLx":
1828, "BLy": 1023, "BRx": 2194, "BRy": 1023, "Label": "", "OCR Length":
0, "OCR Data": "" }


 public boolean checkSignature(int docID, int pImagewidth, int pImageHeight, int
 mMICRy,
 int mDPI, int tLX, int tLY, int tRX, int tRY, int bLX, int bLY, int bRX, int bRY)
 {

  boolean result = false;
  /* processed Image Bitmap */
  Bitmap procImageBitmap = ConstValues.mAppObject.mFrontProcessedBitmap;
  /* create a rectangle image on top of the Image Bitmap received from Metadata*/
Rect mEVRSRect = new Rect();
  mEVRSRect.set(tLX, tLY, bRX, bRY);

  int width = procImageBitmap.getWidth();
  int height = procImageBitmap.getHeight();
  Rect mSignRect = new Rect();
/* Create a rectangle image of ¼ size to the right bottom corner of the image */
  if (docID == 0)
  {
   int Signature_Zone_left = pImagewidth / 2;
   int Signature_Zone_top = pImageHeight / 2;
   int Signature_Zone_right = pImagewidth;
   int Signature_Zone_bottom = pImageHeight;

   mSignRect.set(Signature_Zone_left, Signature_Zone_top, Signature_Zone_right,
   Signature_Zone_bottom);

  }
/* Create a rectangle image of 1/6 size to the top of the Image bitmap */
  else
  {
   mSignRect.set(0, 0, width, height / 6);
```

```
  }
  /*Check for intersection of the rectangles created by Metadata and the ¼ or
  1/6 rectangles*/
  if (mSignRect.contains(tLX, tLY) && mSignRect.contains(tRX, tRY)
  && mSignRect.contains(bLX, bLY) && mSignRect.contains(bRX, bRY))
  {
   result = true;
  }
  return result;
 }


Check for Valid MICR

// we would look for OCR Data and make sure there are 9 digits (0-9) between two C's
present on check front side.

MetaData received after the processing we check for OCR Data key value pair available
in Json format.

{ "Index":    2, "Type": "MP", "TLx":  187, "TLy": 1139, "TRx": 2311, "TRy": 1139,
 "BLx":
187, "BLy": 1217, "BRx": 2311, "BRy": 1217, "Label": "MICR", "OCR Length":   21,
"OCR Data": "C000067895C 12345678P" }
/*Parse the OCRData key and send the value to the function */
/* Function will check for the string containing particular format CXXXXXXXXXC. Where
 X
ranges from 0-9*/
public boolean checkMICR(String OCRData)
 {

  boolean result = false;
  if (!OCRData.equals(""))
  {
   Log.e(TAG, "Raw MICR : " + OCRData);
   OCRData = OCRData.replace(" ", "");
   OCRData = OCRData.toLowerCase(Locale.getDefault());
   if (OCRData.contains("c"))
   {
    int startIndex = OCRData.indexOf('c');
    int endIndex = OCRData.indexOf('c', startIndex + 1);
    if ((endIndex - startIndex) == 10)/* Detects only 9- digits as per
             * US standards */
    {

     Log.e(TAG, "Parsed MICR : " + OCRData);
     String mMICRTxt = OCRData.substring(startIndex + 1, endIndex);
     String expression = "^[0-9]*$";
     Pattern pattern = Pattern.compile(expression);
     Matcher matcher = pattern.matcher(mMICRTxt);
     if (matcher.matches())
     {
      result = true;
     }

    }
   }

  }
  return result;
 }
```

iOS

```
// MetaData we get after processing front of check is as follows

/*
{
"Front Side":
 {
 "Input Image Attributes":
  {
  "Width":2448,
  "Height":3264,
  "Channels":3,
  "BitDepth":24,
  "xDPI":200,
  "yDPI":200
  },
 "Output Image Attributes":
  {
  "Width":2837,
  "Height":1310,
  "Channels":1,
  "BitDepth":1,
  "xDPI":475,
  "yDPI":475
  },
 "Page Detection":
  {
  "Tetragon":
   {
   "Max Deviation from 90 in degrees": 3.425,
   "Rectangularized": true
   }
  },
 "Text Lines":
  {
  "Num": 3,
  "Lines":
   [
   { "Index":    0, "Type": "HP", "TLx": 1828, "TLy":  726, "TRx": 2194, "TRy":  726,
   "BLx": 1828, "BLy": 1023, "BRx": 2194, "BRy": 1023, "Label": "", "OCR Length":
   0, "OCR Data": "" },
   { "Index":    1, "Type": "HP", "TLx": 2204, "TLy":  502, "TRx": 2318, "TRy":  502,
   "BLx": 2204, "BLy":  698, "BRx": 2318, "BRy":  698, "Label": "", "OCR Length":
   0, "OCR Data": "" },
   { "Index":    2, "Type": "MP", "TLx":  187, "TLy": 1139, "TRx": 2311, "TRy": 1139,
   "BLx":  187, "BLy": 1217, "BRx": 2311, "BRy": 1217, "Label": "MICR", "OCR Length":

   21, "OCR Data": "C000067895C 12345678P" }
   ]
  },
 "Auto Orientation":
  {
  "Auto Orientation has been done": true
  }
 }
}

*/

// We look for signature in the bottom right corner rectangle i.e. if check size is
100 x 100
// we would look for human written characters in the rectangle with origin (50,50)
and has width & height of 51

-(void)lookForSignature:(NSString*)metaData{
```

```
   NSError *jsonError;
// converting metadata to a dictionary for easy accessing.
   NSDictionary *jsonDict =  [NSJSONSerialization JSONObjectWithData:
   [metaData dataUsingEncoding:NSUTF8StringEncoding] options:kNilOptions
   error:&jsonError];

if(jsonError != nil){
       return;
   }

   CGPoint BL, BR, TL, TR;
   CGFloat width=0, height=0, xDPI = 0, yDPI = 0;

// Storing details like height, weight, DPI etc of check in local variables

   if([[jsonDict allKeys] containsObject:@"Front Side"]){

       jsonDict = [jsonDict objectForKey:@"Front Side"];

       if([[jsonDict allKeys] containsObject:@"Output Image Attributes"]){

           jsonDict = [jsonDict objectForKey:@"Output Image Attributes"];

           height = [[jsonDict objectForKey:@"Height"] floatValue];
           width = [[jsonDict objectForKey:@"Width"] floatValue];
           xDPI = [[jsonDict objectForKey:@"xDPI"] floatValue];
           yDPI = [[jsonDict objectForKey:@"yDPI"] floatValue];
       }
   }



   CGRect signatureRect = CGRectMake(width/2, height/2, width/2 +1 , height/2 +1 );
   // Forming rectangle whose origin & height and width would look for signature in
   lower right quadrant

   jsonDict = [NSJSONSerialization JSONObjectWithData:[metaData dataUsingEncoding:
   NSUTF8StringEncoding] options:kNilOptions error:&jsonError];

if(jsonError != nil){
       return;
   }

   if([[jsonDict allKeys] containsObject:@"Front Side"]){

       jsonDict = [jsonDict objectForKey:@"Front Side"];

       if([[jsonDict allKeys] containsObject:@"Text Lines"]){

           jsonDict = [jsonDict objectForKey:@"Text Lines"];

           if([[jsonDict allKeys] containsObject:@"Lines"]){

               NSArray *tempArray = [jsonDict objectForKey:@"Lines"];

               for (NSDictionary *tempDict in tempArray) {

                   if([[tempDict valueForKey:@"Type"] isEqualToString:@"HP"]){
                   // Looking for HP or Human Printed characters in Meta Data

                       NSLog(@"HP found");
                       BL = CGPointMake([[tempDict valueForKey:@"BLx"] floatValue],
```

```
                        [[tempDict valueForKey:@"BLy"] floatValue]);
                        BR = CGPointMake([[tempDict valueForKey:@"BRx"] floatValue],
                        [[tempDict valueForKey:@"BRy"] floatValue]);
                        TL = CGPointMake([[tempDict valueForKey:@"TLx"] floatValue],
                        [[tempDict valueForKey:@"TLy"] floatValue]);
                        TR = CGPointMake([[tempDict valueForKey:@"TRx"] floatValue],
                        [[tempDict valueForKey:@"TRy"] floatValue]);


                        if([self search:signatureRect For:BL signature:BR presence:
                        TL AndReturn:TR] == 1){ // This would tell if HP character
                        are inside lower right quadrant or not

                            NSLog(@"SIGNATURE FOUND\n");
                            break;
                        }

                    }
                }

            }
        }
    }

    return;

}

-(int)search:(CGRect)signatureRect For:(CGPoint)BL signature:(CGPoint)BR presence:
(CGPoint)TL AndReturn:(CGPoint)TR{

    if(CGRectContainsPoint(signatureRect, BL) && CGRectContainsPoint(signatureRect,
 BR)&&
     CGRectContainsPoint(signatureRect, TL) && CGRectContainsPoint(signatureRect, TR))
 {

        NSLog(@"signature inside zone 1\n");
        return 1;
    }
    else{

        signatureRect.origin.x -=5;
        signatureRect.origin.y -=5;
        signatureRect.size.height +=5;
        signatureRect.size.width +=5;

        if(CGRectContainsPoint(signatureRect, BL) &&
        CGRectContainsPoint(signatureRect, BR)  &&
           CGRectContainsPoint(signatureRect, TL) &&
           CGRectContainsPoint(signatureRect, TR)) {

            NSLog(@"signature inside zone 2\n");
            return 1;
        }
        else{

            signatureRect.origin.x -=5;
            signatureRect.origin.y -=5;
            signatureRect.size.height +=5;
            signatureRect.size.width +=5;

            if(CGRectContainsPoint(signatureRect, BL) &&
            CGRectContainsPoint(signatureRect, BR)  &&
               CGRectContainsPoint(signatureRect, TL) &&
```

```
                CGRectContainsPoint(signatureRect, TR)) {

                 NSLog(@"signature inside zone 3\n");
                 return 1;
             }
         }
    }

    NSLog(@"signature Outside zone\n");

    return 0;

}


// we would look for OCR Data and make sure there are 9 digits (0-9) between
   two C's present on check front side.
// Sample OCR Data  "OCR Data": "C000067895C 12345678P"


-(void)lookForMICR:(NSString*)metaData{

 NSError *jsonError;
    NSDictionary *jsonDict =  [NSJSONSerialization JSONObjectWithData:
    [metaData dataUsingEncoding:NSUTF8StringEncoding] options:kNilOptions error:
    &jsonError];

    if(jsonError != nil){
        return ;
    }

 if([[jsonDict allKeys] containsObject:@"Front Side"]){

        jsonDict = [jsonDict objectForKey:@"Front Side"];

        if([[jsonDict allKeys] containsObject:@"Text Lines"]){

            jsonDict = [jsonDict objectForKey:@"Text Lines"];

            if([[jsonDict allKeys] containsObject:@"Lines"]){

                NSArray *tempArray = [jsonDict objectForKey:@"Lines"];

                for (NSDictionary *tempDict in tempArray) {

                    if([[tempDict valueForKey:@"Label"] isEqualToString:@"MICR"] &&
                    [[tempDict valueForKey:@"OCR Data"] length] > 0){

                        NSString *ocrData = [tempDict valueForKey:@"OCR Data"];
                        ocrData = [ocrData lowercaseString];
                        NSArray *ocrArray = [ocrData componentsSeparatedByString:@"c"];

                        if([ocrArray count] == 3){


                            ocrData = [ocrArray objectAtIndex:1];
                            ocrData = [ocrData stringByReplacingOccurrencesOfString:
                            @" " withString:@""];

                            if([ocrData length] == 9 && [self isAllDigits:ocrData])
                                break;
                        }
                    }
```

```
                }

            }
        }
    }

}

- (BOOL) isAllDigits:(NSString*)inputString
{
    NSCharacterSet* nonNumbers = [[NSCharacterSet decimalDigitCharacterSet]
 invertedSet];
    NSRange r = [inputString rangeOfCharacterFromSet: nonNumbers];
    return r.location == NSNotFound;
}
```

Target frame cropping

ImageProcessor has a frame pre-cropping functionality. It performs a preliminary crop of the image based on the target frame and can improve page detection by eliminating some background noise. This only works if images are captured with our capture experience or a target frame rectangle object was set on the image manually.

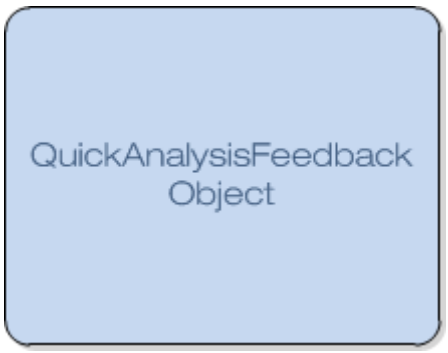
Cropping happens during image processing, prior to page detection. To enable crop to frame:

iOS: Set the `targetFrameCropType` property of `KFXImageProcessorConfiguration` to `TARGET_FRAME_CROP_ON`.

Android: Set the `targetFrameCropType` property of `ImageProcessorConfiguration` to `TargetFrameCropType.TARGET_FRAME_CROP_ON`.

QuickAnalysisFeedback object

The QuickAnalysisFeedback object contains the results of image quality checking using the `doQuickAnalysis` method of the ImageProcessor object.



QuickAnalysisFeedback Object Diagram

The `doQuickAnalysis` method performs a variety of image quality checks for the Image objects, populates the `imageQuickAnalysisFeedback` property in the input `Image`, and fires the `analysisCompleteEvent`.

The `analysisCompleteEvent` returns a status code, info/error status message string, and the Image object reference. The Image object parameter references the original input image that was passed to `doQuickAnalysis` originally. This Image will have the `imageQuickAnalysisFeedback` property populated.

The application can read the following properties of the QuickAnalysisFeedback object:

- `isOversaturated`: true if the image is over saturated
- `isBlurry`: true if the image is blurry
- `isUndersaturated`: true if the image is under saturated
- `tetragonCorners`: found coordinates of the bounding tetragon
- `viewBoundariesImage`: native Bitmap object; shows green line drawn around detected page edges. Acts as a visual representation of found `tetragonCorners` property.

OpenCV

If you need to use OpenCV in your application, please take note of the following information.

Android

When using OpenCV with Android:

- Download `OpenCV-3.1.0-android-sdk.zip` from the repository (`https://github.com/Itseez/opencv/releases`).
- Replace `libopencv_java3.so` in the Mobile SDK with the similar binary from the related release folder (`OpenCV-android-sdk/sdk/native/libs`).

iOS

Since, OpenCV 3.1 (imgproc and core modules) is built into the Mobile SDK on iOS, there is no simple way to replace it. If you want to use the imgproc and core modules from OpenCV, you can simply rely on the Mobile SDK. Additional linking with OpenCV is not needed. Just update the project file with header files from OpenCV.

If you want to use additional modules from OpenCV, you will have to build and link them to your project.

Here are the basic steps:

1. Download the OpenCV source code from the repository (`https://github.com/Itseez/opencv/releases`).
2. Update build script to specify required modules: `/<your_workspace>/opencv/platforms/ios/build_framework.py`.
   See the following code snippet for an example.

```
<<<
        for t in self.targets:
            mainBD = self.getBD(mainWD, t)
            dirs.append(mainBD)
            cmake_flags = []
            if self.contrib:
```

```
                cmake_flags.append("-DOPENCV_EXTRA_MODULES_PATH=%s" %
self.contrib)
            if xcode_ver >= 7 and t[1] == 'iPhoneOS':
                cmake_flags.append("-DCMAKE_C_FLAGS=-fembed-bitcode")
                cmake_flags.append("-DCMAKE_CXX_FLAGS=-fembed-bitcode")
            cmake_flags.append("-DBUILD_opencv_core=ON")
            cmake_flags.append("-DBUILD_opencv_imgproc=ON")
            cmake_flags.append("-DBUILD_opencv_world=OFF")
            self.buildOne(t[0], t[1], mainBD, cmake_flags)
            self.mergeLibs(mainBD)
        self.makeFramework(outdir, dirs)
>>>
```

3. Follow the readme on how to build OpenCV: `/<your_workspace>/opencv/platforms/ios/readme.txt`
4. A successful build with result in OpenCV binaries under: `/<your_workspace>/ios`.
5. Create a fat library to built all the architectures into a single binary using the "libtool" command. Here is an example of the `.sh` file:

   ```
   $DT_TOOLCHAIN_DIR/usr/bin/libtool" -static "/<your_workspace>/ios/build/
   <module>_arm64.a" "/<your_workspace>/ios/build/<module>_armv7.a" "/
   <your_workspace>/ios/build/<module>_i386.a" "/<your_workspace>/ios/build/
   <module>_x86_64.a" -o "/<your_workspace>/<module>_universal.a
   ```

6. Link the extra OpenCV module from Step 5 to your xCode project and add the required header files from OpenCV.

Server objects

Can be found in Logistics.

the following sections provide information about the server objects in the Logistics library.

Capture Server

`CaptureServer` is a new class in the Logistics part of the Mobile SDK, which helps in communicating with `FrontOfficeServer` and `TotalAgilityServer`

This capture server class provides a number of methods to facilitate the interface with an external server.

- `registerDevice`: Register a mobile device. This method must precede any of the following methods. A successful device registration is sufficient for multiple login/logout operations.
- `login`: Log in to the server associated with the URL provided by the application by using username and password. After a successful login, the `documetTypes` list will be returned.
- `loginAnonymously`: Log in to the server associated with the URL provided by the application as a guest user. After successful login, the default `documetTypes` list will be returned
- `logout`: Log out from a server.
- `getDocumentType`: Get a specified document type.
- `submitDocument`: Submit a Document object to a server.
- `sendImageService`: Send an image to this server.
- `startJobService`: Start a new Document submission on the server.

Before performing any of the above methods, a valid license must be set.

Examples

```
String serverUrl = "http://xxx.xx.xx.xx/TotalAgility/kofax/kfs/legacy/ws/";
```

registerDevice API for Android

```
CaptureServer ktaServer=new CaptureServer(serverUrl,
 DocumentType.SourceServer.SERVER_KTA);
ktaServer.registerDevice(new ICompletionListener<Void>() {
    @Override
    public void onComplete(Void aVoid, Exception e) {
        if (e!=null) {
            Log.e("Exception",e.getMessage());
        } else {
            Log.i("registerDevice ", "success");
        }
    }
});
```

registerDevice API for iOS

```
NSString *serverUrl = "http://xxx.xx.xx.xx/TotalAgility/kofax/kfs/legacy/ws/";
KFXCaptureServer* server = [[KFXCaptureServer alloc] initWithType:KLO_SERVER_KFS URL :

                           serverUrl];
[server registerDevice : ^(int responseCode, NSError* error) {
 if(error != nil){
   NSLog(@"error occurred %@",error.localizedDescription);
 }
 else{
   //successfully registered
 }
}];
```

login API for Android

```
CaptureServer ktaServer=new CaptureServer(serverUrl,
 DocumentType.SourceServer.SERVER_KTA);
UserProfile  userProfile = new UserProfile();
userProfile.setDomain(domain);
userProfile.setUsername(userName);
userProfile.setPassword(password);
userProfile.setUserEmailAddress(email);
ktaServer.login(userProfile, new ICompletionListener<List<String>>() {
    @Override
    public void onComplete(List<String> strings, Exception e) {
        if (e != null) {
            Log.e("Exception", e.getMessage());
        } else {
            Log.i("Login ", "Success");
        }

    }
});
```

login API for iOS

```
KFXCaptureServer* server = [[KFXCaptureServer alloc]
    initWithType:KLO_SERVER_KFS URL:@"serverurl"];
    kfxKLOUserProfile* userProfile = [[kfxKLOUserProfile alloc] init];
[userProfile setDomain:@"domain"];
[userProfile setUsername:@"username"];
[userProfile setPassword:@"password"];
[userProfile setUserEmailAddress:@"email"];

[server login:userProfile completionHandler:^(int responseCode, NSArray
    *documentTypeNames, NSError* error){

        if(error != nil){
            NSLog(@"%@", error.localizedDescription);
        }
        else{
            //save document types
        }

}];
```

loginAnonymously API for Android

```
CaptureServer ktaServer=new CaptureServer(serverUrl,
 DocumentType.SourceServer.SERVER_KTA);
ktaServer.loginAnonymously(new ICompletionListener<List<String>>() {
    @Override
    public void onComplete(List<String> strings, Exception e) {
        if (e != null) {
            Log.e("Exception", e.getMessage());
        } else {
            Log.i("Anonymous Login ", "Success");
        }
    }
});
```

loginAnonymously API for iOS

```
KFXCaptureServer* server = [[KFXCaptureServer alloc]
        initWithType:KLO_SERVER_KFS URL:@"serverurl"];
[server loginAnonymously:^(int responseCode, NSArray *documentTypeNames, NSError*
 error){
        if(error != nil){
            NSLog(@"%@", error.localizedDescription);
        }
        else{
            //save document types
        }
}];
```

logout API for Android

```
CaptureServer ktaServer=new CaptureServer(serverUrl,
 DocumentType.SourceServer.SERVER_KTA);
ktaServer.logout(new ICompletionListener<List<String>>() {
    @Override
    public void onComplete(Void aVoid, Exception e) {
```

```
        if (e != null) {
            Log.e("Exception", e.getMessage());
        } else {
            Log.i("logout ", "Success");
        }
    }
});
```

logout API for iOS

```
[server logout:^(int responseCode, NSError* error){
        if(error != nil){
            NSLog(@"%@", error.localizedDescription);
        }
        else{
            //log out successful
        }
}];
```

getDocumentType API for Android

```
CaptureServer ktaServer=new CaptureServer(serverUrl,
 DocumentType.SourceServer.SERVER_KTA);
ktaServer.getDocumentType(documentTypeName, new ICompletionListener<List<String>>() {
    @Override
    public void onComplete(DocumentType documentType, Exception e) {
        if (e != null) {
            Log.e("Exception", e.getMessage());
        } else {
            Log.i("getDocumentType", "Success");
        }

    }
});
```

getDocumentType API for iOS

```
[server getDocumentType:documentType completionHandler:^(int responseCode,
        kfxKLODocumentType* documentTypeObject,NSError* error){
        if(error != nil){
            NSLog(@"%@",error.localizedDescription);
        }
        else{
            //get documenttype success
        }
}];
```

submitDocument API for Android

```
CaptureServer ktaServer=new CaptureServer(serverUrl,
    DocumentType.SourceServer.SERVER_KTA);
ktaServer.submitDocument(document, progressListener,
    new ICompletionListener<List<String>>() {
    @Override
    public void onComplete(int successCode, Exception e) {
        if (e != null) {
            Log.e("Exception", e.getMessage());
        } else {
```

```
            Log.i("submitDocument ", "Success");
        }

    }
});
```

submitDocument API for iOS

```
[server submitDocument:DOCUMENT completionHandler:^(int responseCode,NSError* error){

        if(error != nil){
            NSLog(@"%@", error.localizedDescription);
        }
        else{
            //Submit document complete
        }
    }progressHandler:^(int progress,NSString* submissionJobId,NSError* error){
        // progress contains the status
 }];
```

sendImageService API for Android

```
CaptureServer ktaServer=new CaptureServer(serverUrl,
 DocumentType.SourceServer.SERVER_KTA);
ktaServer.sendImageService(document, jobId, image, imageIndex, isLastImage,
    new ICompletionListener<List<String>>() {
    @Override
    public void onComplete(int successCode, Exception e) {
        if (e != null) {
            Log.e("Exception", e.getMessage());
        } else {
            Log.i("sendImageService ", "Success");
        }

    }
});
```

sendImageService API for iOS

```
[server sendImageService:document image:image jobID:jobid imageIndex:imageIndex
 lastImage:
        NO completionHandler:^(int responseCode,NSError* error){
        if(error != nil){
            NSLog(@"%@", error.localizedDescription);
        }
        else{
            //sendImageservice complete
        }
 }];
```

startJobService API for Android

```
CaptureServer ktaServer=new CaptureServer(serverUrl,
 DocumentType.SourceServer.SERVER_KTA);
ktaServer.startJobService(new ICompletionListener<List<String>>() {
    @Override
    public void onComplete(String jobId, Exception e) {
        if (e != null) {
```

```
            Log.e("Exception", e.getMessage());
        } else {
            Log.i("JobID", jobId);
        }

    }
});
```

startJobService API for iOS

```
[server startJobService:^(NSString* jobID, int responseCode,NSError* error){
        if(error != nil){
            NSLog(@"%@", error.localizedDescription);
        }
        else{
            //start job service complete
        }
}];
```

Server extraction objects

`ServerExtractor` is a new class in Logistics part of the Mobile SDK, which helps in communicating with Smart Mobile Components in SMC projects such as Kofax Mobile Deposit Capture, Kofax Mobile Bill Pay and Kofax Mobile ID Capture. This class can even be used to send a multi-part request to any server, which supports multi-part extraction of images. This class is optimized to work with Kofax Smart Mobile Projects.

This class exposes APIs to extract data from valid documents (images) like IDs, Checks and Bills. Images can be either in memory (`bitmap`) or on a disk (`filepath`). It returns a JSON received from server or error code in case of an error. The error codes and description received from Server in general is not parsed and returned as is.

Before using these classes it is recommended to look at the SMC project documentation to become familiar with the expected parameters for a typical request required to extract data from images.

Using server extractor

There are two major server installations Real-Time Transformation Interface and Kofax TotalAgility in which the SMC projects can be used for extraction. There are two server objects `RTTIServerConnection` and `KTAServerConnection` that are provided by the SDK, which represent the above, said installations. These server objects can be instantiated with the destination URL. A time out for the connection can also be set on these server objects. In case of `KTAServerConnection` a login method is exposed to explicitly login to Kofax TotalAgility in server and get the required `documentid`, `sessionid` or any other required parameters.

`ServerExtractor` class needs to be instantiated with either one of the above server objects to perform extraction.

Examples

iOS:

```
KFXRTTIServerConnection* rttiServer = [[KFXRTTIServerConnection alloc] initWithURL:
```

```
   @"https://mobiledemo.kofax.com"];
KFXServerExtractionParameters* parameters = [[KFXServerExtractionParameters alloc]
   initWithImages:processedImages];
parameters.parameters = params;

kfxKLOServerExtractor* serverExtractor = [[kfxKLOServerExtractor alloc]
    initWithConnection:customServer];
serverExtractor.delegate = self;
[serverExtractor extract : parameters];
```

Android:

```
IServerExtractor serverExtractor = ServerBuilder.build(getApplicationContext(),
   ServerBuilder.ServerType.RTTI);
 serverExtractor.extractData(serverExtractionParameters,
   new ICompletionListener() { @Override public void onComplete(String response,
   Exception e) { } });
IServerExtractor serverExtractor = ServerBuilder.build(getApplicationContext(),
   ServerBuilder.ServerType.KTA);
serverExtractor.extractData(parameters, new ICompletionListener() {
   @Override public void onComplete(String response, Exception e) {} });
```

Extraction parameters contains the images to be extracted and a dictionary of server input parameters like `xregion` or `processimage = true`, etc. These are essentially the server flags required as part of the request to perform extraction. If the parameters are not valid, or if the required parameters are not sent, extraction may fail.

The `KFXServerExtractorDelegate` contains the `callbacks`, which return the extracted results and errors in case of errors.

Kofax Front Office Server logon

An application can establish a connection with Kofax Front Office Server, making it possible to interact with it via SSL as a supported security method. `Connection` and `Session` properties can be "set" or "get."

When using this object, there are methods that make it possible to log in or log out from the server and register the mobile device with the server.

TotalAgility Server logon

An application can establish a connection with TotalAgility, making it possible to interact with it via SSL as a supported security method. `Connection` and `Session` properties can be "set" or "get."

TotalAgility Server interface

The Kofax Front Office Server and TotalAgility Server objects provide a number of methods to facilitate the interface with an external server:

- `registerDevice`: Register a mobile device. This method must precede any of the following methods. A successful device registration is sufficient for multiple login/logout operations.
- `login`: Login to the server associated with the URL provided by the application.
- `logout`: Logout from a server.
- `cancel`: Cancel a previously started `submitDocument` method.

- `getDocumentTypeList`: Get a list of available document types.
- `getDocumentType`: Get a specified document type.
- `submitDocument`: Submit a Document object to a server.

Before performing any of the above methods, a valid license must be established.

All of the above server interface methods operate asynchronously, except for `getDocumentTypeList` which is synchronous. That is, each asynchronous method initiates the server interface communication and returns immediately to the application. The final success or failure of server interface methods is returned to the application in an event. If a failure occurs, the event includes an error code indicating the specific failure status.

Logging into a server

In order to log into a server, the application must first specify the necessary credentials.

To specify credentials, the application must create a User Profile object and set the username, password, optional email address and optional domain. It can initialize one or more of these user objects as required, and associate one with a Kofax server object. The Kofax server object will indicate if any of these user data properties are invalid when they are used.

The application may not necessarily need to log into a server to perform library operations. Therefore, it may login at any time when necessary to establish a remote connection.

To log into a Kofax server, the application must first set a valid license. Following that, the application must use the appropriate Kofax Front Office Server or TotalAgility server class, supply user credentials and the server URL, and call the register method followed by the login method. (The register method need only be called once for a given server. Multiple login and logout operations can be performed for a single register.) If the connection fails, the error code in the Server object will indicate the reason for the failure. The application is responsible for presenting Server object error messages to the user.

The library responds with an asynchronous event when login completes, indicating success or failure. The library sets the error code in the Server object to indicate why login failed (bad or expired license, no WiFi, bad username, bad password, already logged into another server, or login cancelled). The application should always display error messages for these errors.

Upon successfully logging in, the library stores available document type names which can be accessed with the `getDocumentTypes` method. The document types are only valid and available if the login event indicates success. If the login failed, the `getDocumentTypes` method returns a null pointer, and all other methods similarly set an error code to indicate that it is not connected to the server.

The library will detect this condition and not attempt to perform the Server object method. Instead, it will return the appropriate null data for these calls. Therefore, the application should always check for null pointers and error codes when the event indicates failure.

Indicated errors

The library will send an event to indicate that login failed. In addition, the library times out after approximately 60 seconds, and returns a login failed event.

Before attempting login, the library checks the following:

- If the license is valid: the error code indicates that the license is not valid

- If the user is already logged in: The method event indicates success, and the error code is set to "Already Logged in."
- That the user is not in the process of logging out : If so, it means the application did not wait for logout to complete, and for this case the library returns an overlap error code.
- WiFi access: If none, the error code indicates that WiFi is not available.
- That all available document types are received: The library will return an error if WiFi service is lost or some other condition occurred.
- That no other Kofax Server object exists with a session status indicating that the application is already logged into another server. The library returns an error code for this case.

Login cancellation

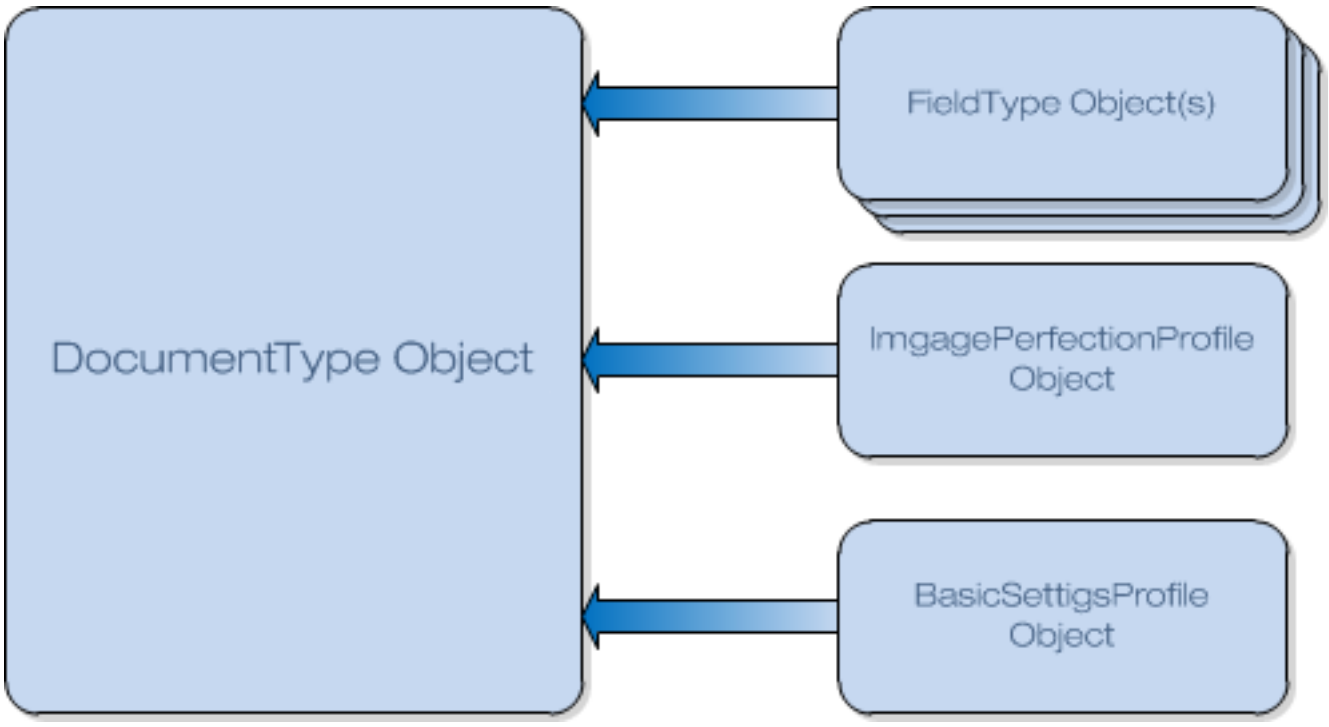
Application users can cancel a login operation at any time using a cancel button or some other control. The action for this control calls the cancel method on the Server object. The library will attempt to cancel the login, and indicate in the error code that the login was cancelled.

If the application calls the cancel method at about the same time the login completes successfully, the library will honor the cancel, and logout from the server if necessary. If more than 10 seconds have elapsed after a successful login, the library ignores the cancel request. This protects against a potential race condition.

DocumentType object

Can be found in Logistics.

The DocumentType object is a model that describes the attributes of a document. A DocumentType object can be defined at design time or obtained at run time from a server such as the Kofax Front Office Server or TotalAgility. The DocumentType object will include only one active object either a `BasicSettingsProfile` or an `ImagePerfectionProfile` object to define the type of image processing to perform for this type of document.



DocumentType Object

The DocumentType object is instantiated with an array of `FieldType` which becomes the `fieldTypes` read-only property. Once instantiated, the `fieldTypes` property cannot be modified by the application.

Once the application has logged in to a server it can access the available document type names on the server by calling the `getDocumentTypes` method on the Server object. This call returns an array of strings for names of documents that could possibly be created. The application must use one of these name strings, without modification, to create a Document object.

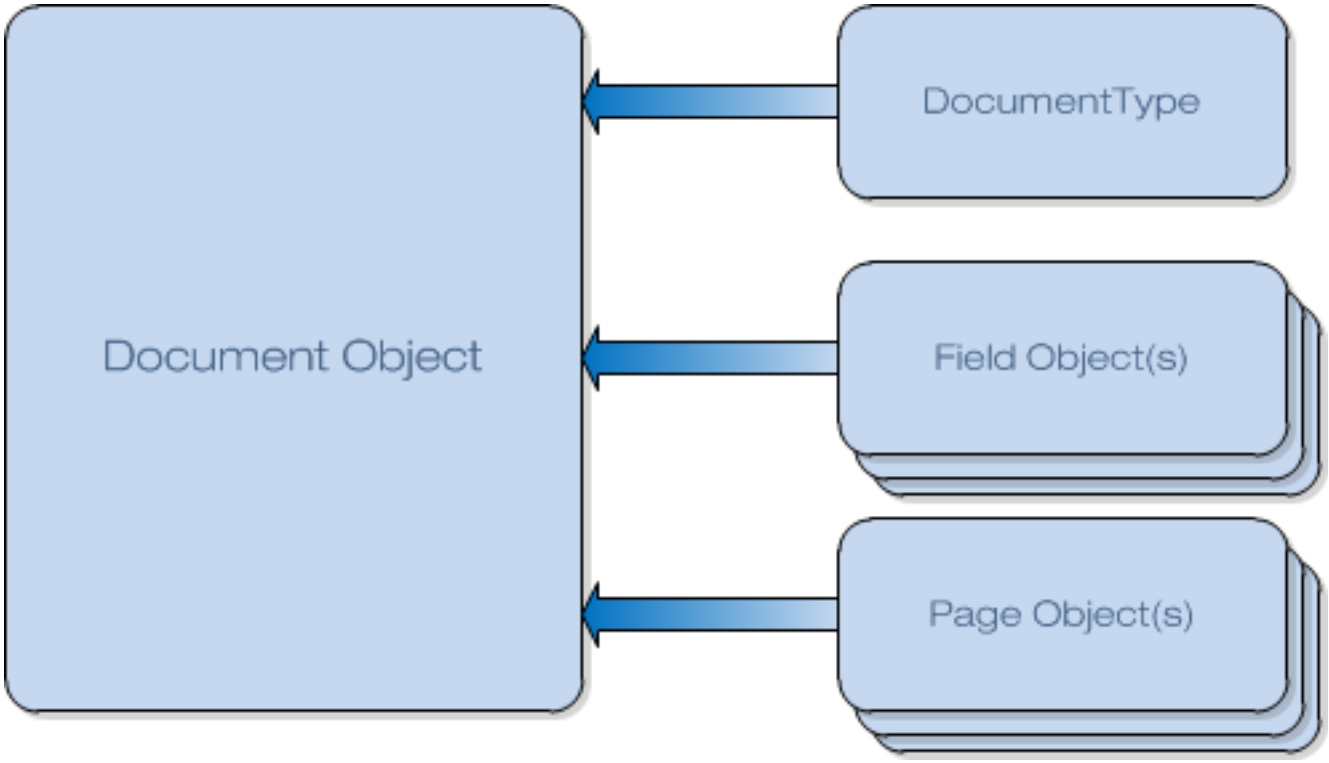
A valid document type is required to create a Document object. The Document object is a place to add pages. A Document object is also required to submit the document to a Kofax Server or a custom-built server.

The library will return an error to this method if the application is not logged in.

There is no need to process the `DocumentType` name strings returned from the library, because the library removes any and all escape sequences, such as %20 for the space character.

Document object

The Document object is instantiated as a specific `DocumentType`. A Document object contains zero or more Field objects and one or more Page objects

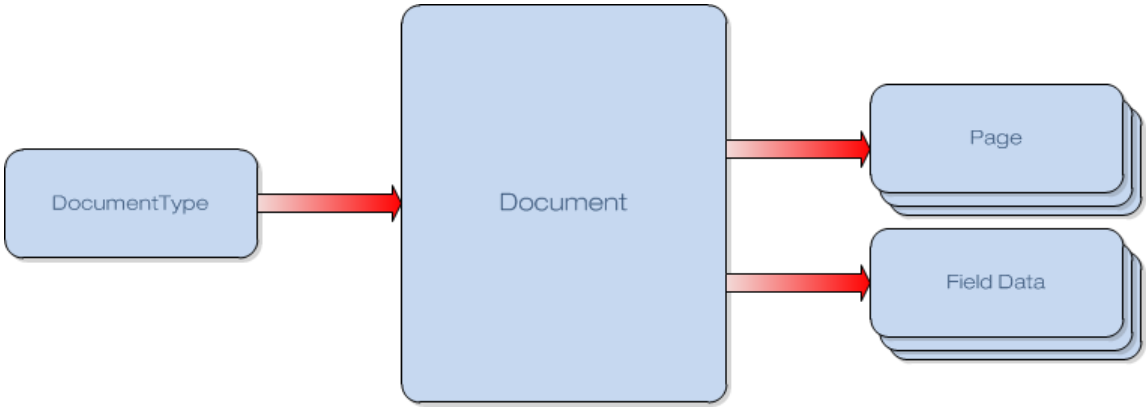


Document Object

At instantiation time, the fields property is created by examining the `fieldTypes` property of the `documentType` property, and instantiating a new Field object for each `FieldType` in the `fieldTypes` array. These Field objects are placed in an array, and made available to the application.

The document class constructor creates the reference to the DocumentType object used to create the document object, along with an uninitialized array with a single empty page. It also sets the other document properties to their default settings. When working with images intended for the Kofax Front Office Server or TotalAgility server, the application works with a document object to prepare it for submission.

The application must create a Document object in order to begin or continue adding pages to the document.



Saving a Document Object

To create a Document object, instantiate a document class by providing a document type object as a parameter. The library initializes and generates the field data array from the field type array in the DocumentType object. The current value in each field comes from the DocumentType object. Then the application can fill in the Field objects as needed, either manually or in combination with extraction or from some user interface screen.

Since the application creates the Document object itself, no errors are returned or expected. Also, the library does not generate an event. The Document object constructor returns a Document object.

All document objects contain a DocumentType object reference. This reference must remain current during the life of the Document object because it contains information necessary for controlling document processing, data verification and submission. If it becomes necessary to restore the fields array with default values, the application can copy the field type data to the field data array.

The application can save the Document object to disk and open it later to continue adding pages as needed. Multiple documents can be open at the same time, but the library can work with only one document at a time.

An application that uses the Document object controls the flow of image processing. The SDK provides the Document object as a container for images. The Document object contains all the document pages and associated data. You add pages to the document using the `addPage` method. You add images to a page using the `addImage` method on the page. You can create images by using the UIControl objects or by using an output image from the image processor. Then you can add these images by using the `addImage` method in your application.

Note The library does not fetch all document fields for all document types in the background. That is the responsibility of the application.

The application can also initialize and use a DocumentType object of its own design.

Page object

A page represents the front or back side of an original sheet. A page can have multiple images, but only one can be marked as the current image. This way the application can maintain the original, and restore the original as the current page if desired.

Conversely, the application could destroy the original image, and set the processed image as the current. Or, after reviewing the image, it could delete the image altogether (using the `RemoveImage` on the Page object) and repeat image capture. It is up to the application to decide image management policy. The code must indicate the current image for that page, by setting an array index in the Page object, the first image having an index of 0.

When there are multiple images within a Page object, these images could be the original unprocessed image, a partially-processed image , an intermediate image, the final image, or the current image. Image objects can be freely deleted, keeping in mind that the raw image takes up a large amount of the available memory. It may be necessary to store that image to disk, and restore it to the image buffer as needed.

The application manages these images and sets the `currentPageIndex` to mark the image that will be submitted to the server.

If `currentPageIndex` is applied to an image that doesn't exist in the page array, the library will use the last image in the array to avoid a fatal error. However, the application should make every attempt to set a valid image index. There must be at least one page with one image to submit a document to the server, otherwise the library returns an error with the submit method. The library detects and reports this error.

FieldType object

The FieldType object describes the attributes of a Field object.

In order to get field data, the application must create a DocumentType object using one of the document type names. Then, call the library to initialize the balance of the DocumentType object. Upon calling the `getDocumentType` method the library will:

- Fetch the document field types and fill in the array of field types in the DocumentType object, if that DocumentType object field type array is currently empty. The field type array will include all fields, and each field contains things like the field name, min, max, and the current (default) setting.
- Populate the basic settings or image perfection profile associated with this document type.
- Return a `kfsDocTypeResultEvent` event, with the DocumentType object and an error code when there is a failure.

Field object

The Field object is instantiated using a particular FieldType object and contains a value which conforms to that Field Type.

Each Field object in the current Document object has a property to store all the possible extracted data options for that field in an array. The first item is the most probable, and this is stored in the field value. The other values stored in the classification results array are possible alternates.

When validating fields, upon completion of the Web services exchange, the library will generate an event to notify the application that the validation request has been completed. The event will indicate if the validation completed with or without error.

If the validation completed with an error, then inspect the `fieldsAreValid` bool setting of the Document object to locate invalid fields. If the fields are invalid, then the application can iterate through the Field objects associated with the document.

When an invalid Field object is found, inspect the `errorDescription` string in the Field object and possibly raise a pop up.

Note The constructor for the Field object should set all fields to invalid. Then when document verification is executed, the Valid flag is set if valid. This forces an application user to go through verification.

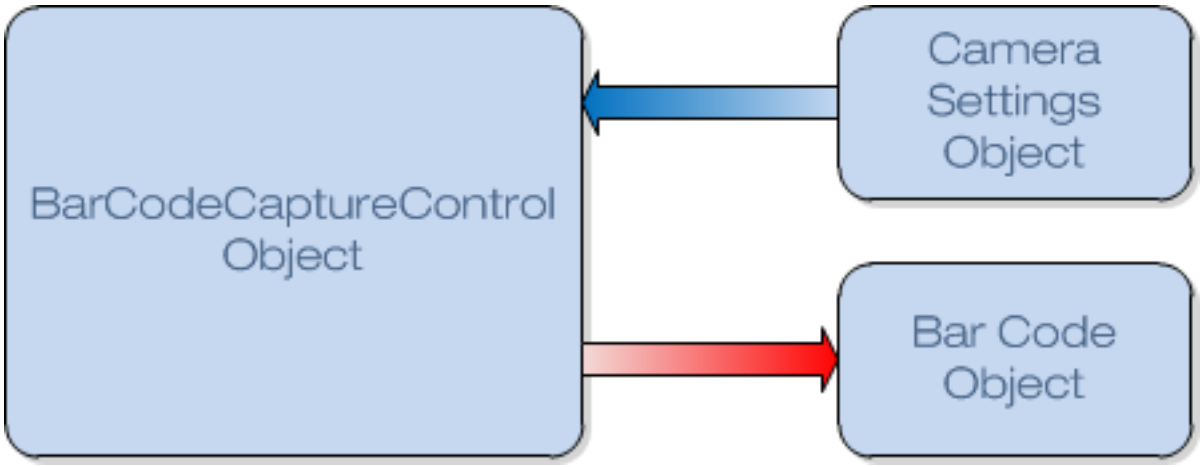
The server performing the validation generates and downloads this string when a validation fails and the library stores this string in the Field object. There is no error code generated by the field validation script, only an error string.

Depending on the error code, the application must iterate through the field objects to determine what field has errors, and carry out appropriate corrective actions.

BarCodeCaptureControl object

Can be found in UIControls.

The BarCodeCaptureControl object shows a camera preview window that continuously sends low-resolution preview images to the image processor, where the bar code reading module scans for bar codes. Features include user feedback cues to control levelness, camera shake, and bar code positioning. The application is responsible for creating and displaying any screen overlays that offer user feedback, such as "Hold camera still."



BarCodeCaptureControl Object Diagram

If the application needs to read one bar code, and that is likely the only thing to be processed, then it can use the BarCodeCaptureControl object, which has methods to highlight the bar code, capture an image and capture the data from it.

The BarCodeCaptureControl object asks the library to capture and return an unprocessed image of a bar code and its bar code data.

To capture a bar code, the application instantiates the object and sets the desired capture feedback features to help users take clear pictures. The application also specifies the type of bar code to recognize.

The stability and orientation thresholds help ensure the camera is level and steady before the library captures an image.

To start the process of capturing a bar code, the application calls the `readBarCode` method.

As soon as the application gets the event for the bar code, the library stops the feedback events and the application can process that bar code data. The bar code data includes the location of the bar code in the image, the bar code type, and the bar code contents and length.

The library generated image need not necessarily be associated with a page. The image and bar code data can later be added to a page with `addImage`.

Supported bar codes for BarCodeCaptureControl object

The following bar codes and variants are supported:

Note All UPC codes are also EAN codes (EAN is a superset of UPC). If only the EAN symbology is selected, then UPC codes will be reported as EAN.

- CODE93
- DATAMATRIX
- CODE39
- EAN
  - EAN13
  - EAN8
- UPC
  - UPCA
  - UPCE
- CODE128
- PDF417
- QR
- AZTEC
- CODE25
  - 2 of 5 interleaved
  - 2 of 5 standard
- CODABAR

Reading techniques

There are two types of bar code reading techniques in the library. When the application needs to read and process only one bar code, use the BarCodeCapture object with `maxBarCodes` set to 1. In this case, there are user interface methods to guide the user through the process of highlighting the bar code, capturing the image, and capture the data from the bar code.

Note The BarCodeCaptureControl object only finds bar codes, and does no other image processing.

The BarCodeCaptureControl object supports the following search directions:

- All
- Horizontal
- Vertical

Bar code reader and return all bar codes

For those cases where the application needs to read and process multiple bar codes on a single page, the SDK provides a bar code reader separate from the bar code UI control that can be used for processing images. Set `maxBarCodes` set to the maximum number of bar codes you want found.

This method finds all the bar codes on the sheet, where you can specify different orientations and bar code types. There is no UI guidance in this case.

The application uses the bar code reader to scan an arbitrary image for one or more bar codes and populate its internal bar code list with the results. By requiring bar code data to include bound, position, and direction information, any image processing that occurs later can use the results to do special processing or avoid destructive transforms on those areas.

Given a set of parameters, the bar code reader will return all the bar codes it can find in the image. For each bar code that is returned, the following information is included:

- type
- value
- bounding area
- directions

The bar code reader supports the following search directions:

- All
- LEFT_RIGHT
- RIGHT_LEFT
- TOP_DOWN

Supported bar codes for bar code reader

The following bar codes and variants are supported:

Note All UPC codes are also EAN codes (EAN is a superset of UPC). If only the EAN symbology is selected, then UPC codes will be reported as EAN.

- CODE93
- POSTNET
- CODE39
- EAN
  - EAN13
  - EAN8
- UPC
  - UPCA

    - UPCE
- CODE128
- PDF417
- QR
- CODE25
    - 2 of 5 interleaved
    - 2 of 5 standard
- CODABAR

BarcodeReader object

The BarcodeReader object is a stand-alone asynchronous bar code reader that can search for multiple bar codes and multiple bar code types in an image with just one call of the `readBarcodes` method. Unlike the bar code capture control, it has no UI element to it, and will add the results found to the image passed into it.

It has the following three properties:

- An array of bar code types that can be searched for.
- The maximum total number of bar codes that could possibly be returned.
- The search direction

These three parameters must be set in order to set up the bar code reader object before calling `readBarcodes`. Defaults for these properties are: an empty array of bar code types, the maximum number of bar codes to be found is set to one, and "All" for the search directions.

The bar code results that get added to the image after `readBarcodes` is called are of the same type as the bar code capture control would return, so a bounding box, data format, direction, type, and bar code value get added for each one of the bar codes found by the BarcodeReader object. The results and image are returned in the read complete event, or call back method for java, and iOS respectively.

Note As a best practice, try to specify the absolute minimum number of bar codes and bar code types the application should search for and return. Otherwise, in some cases multiple results with bogus data may be returned.

Reading techniques

Take care to use images that are in focus. Images that are overly dark or out of focus will result in poor bar code reading results, or no results at all.

Note The BarcodeReader object only searches for bar codes, and does not perform any other image processing on the image.

Supported bar code types

The following bar code types and their variants are supported.

Note All UPC codes are also EAN codes (EAN is a superset of UPC). If only the EAN symbology is selected, then UPC codes will be reported as EAN.

- CODE128
- EAN
  - EAN13
  - EAN8
- UPC
  - UPCA
  - UPCE
- CODE39
- CODE25
- CODABAR
- PDF417
- QR

Guideline feature

The BarCodeCapture object has a real-time view of the sheet that guides the user to properly take an image of a bar code. A sheet is a representation of the printed original piece of paper, which has a front and back page.

To capture a bar code, the application instantiates the BarCodeCapture object and sets the capture feedback features to use. One feedback feature is a guideline that the user can align through the approximate center of the target bar code. The guideline helps the user position the camera relative to the bar code to facilitate capturing a readable image.

Before the library takes the picture of the bar code, the application can enable (or disable) the guideline. When the guideline is enabled and the view is instantiated, the library opens up a camera view with a color guideline option instead of a finder frame.

The purpose of the Bar Code object is to capture and return an unprocessed image of a bar code and its data. This object captures and generates a unique bar code output image with one bar code.

The Bar Code Capture object has an associated real-time view of the sheet, and helps users position the camera by means of a guideline. To capture a bar code using a guideline, first instantiate the Bar Code Capture object, and set the capture desired feedback features. Also specify the type of bar code to recognize. Next enable the feedback guideline that helps users position the camera so the guideline passes through the target bar code.

To start the process of capturing a bar code, call the `readBarCode` method. This starts the flow of periodic feedback events, and after the feedback thresholds are met, the library calls the `BarCodeFound` event. This event gives the application the Image object, including the bar code data stored in that Image object.

As soon as the application gets the event for the bar code, the library stops providing feedback events, and the application can process that bar code data as appropriate. The bar code data includes the location of the bar code in the image, the bar code type, and the associated bar code contents and length.

The library generated image need not be associated with a page. At a later time in the process the application can add the image with the bar code to a page using `addImage`.

License capture control object

Can be found in UIControls.

The License Capture Control object is a smart camera view that detects and recognizes the Mobile SDK software license in the QR code format.

Once the license is found, it will be used for Mobile SDK licensing. Features include screen overlays that offer user feedback, such as instruction messages and a static frame. Properties of this object allow customizing the location and size of the control within the app's user interface.

In order to use the License Capture Control, it is not necessary for the Mobile SDK is to be licensed.

To license the Mobile SDK via the License Capture Control, the application instantiates the object (Android: `LicenseCaptureView`, iOS: `kfxKUILicenseCaptureControl`), and sets a listener for a capture event. Optionally, the application can also specify a user instruction message, the static frame color, and padding.

To start the process of capturing a license, the application calls the `readLicense` method.

As soon as the application gets the event for the license, the library stops the feedback events, and the application can process that license data. The license data includes the status code of the Mobile SDK license. If licensing was successful, it will also include the license string and the number of days remaining until the license expires.

Note A different API is used for manually entering the SDK license key. See Using the SDK with iOS and Using the SDK with Android for examples and/or guidance.

Credit card capture

The Credit card capture control should be used only for embossed credit cards only. It does not support nom-embossed credit cards.

When capturing embossed credit cards, a dedicated capture control (`CreditCardCapture`) is used which presents a dedicated interface. This interface captures the card and extracts the card number data. An event is then fired that returns a credit card info object.

Note The SDK does not store credit card information for any period of time beyond that necessary to process and extract the credit card number and expose it to the application developer.

Expiration dates

The SDK credit card scanner can extract expiration dates from images of credit cards. However, there are some known limitations

- Expiration date scans will sometimes fail to obtain the correct date value. Proper illumination of the credit card is essential; strong lighting seems to work best. Different lighting angles may also help. If there is not enough light, the expiration date will not be extracted.
- The credit card scanner returns an expiration date only if it finds one no older than the current month/year, and no more than 5 years into the future.
- Only MM/YY and YY/MM formats are extracted. Cards made with date formats including a specific day of the month, such as 09/22/15 will not have the date extracted. Further, cards made with date formats including a 4-digit year, such as 09/2015, will not have the date extracted.

SDK Version object

Can be found in Utilities.

This object is only used with the SDK Utility package to get the version of the SDK. It also provides access to the version information of the Utility package itself and other components used in the SDK.

Version object

Use the Version object to retrieve version information for an SDK package. Each of the following packages contains a Version object including their unique version information:

- UI Control Objects package
- Logistics Objects package
- Engine Objects package

UI control objects package

Version information for the UI Control Objects package is maintained in the KUIVersion object.

Logistics objects package

Version information for the Logistics Objects package is maintained in the KLOVersion singleton object.

Engine objects package

Version information for the Engine Objects package is maintained in the KENVersion singleton object.

App Statistics overview

Can be found in Utilities.

App Statistics is a feature of the mobile SDK that facilitates recording performance metrics on a device. The application periodically may export the data, and upload it to a server for collection and further analysis. The goal of collecting these statistics is to tell:

- Which functions of the SDK are being used
- Where end users are having difficulty using the SDK functions
- How long SDK functions are taking
- What errors users are experiencing when using the SDK
- What device is generating the data, its operating system, and other device properties

If the application also incorporates the Real-Time Transformation Interface server, then the device-side statistics that are collected can be uploaded and combined with the server side metrics to provide a complete, end-to-end traceability of performance and usage metrics.

Application Statistics collected on the mobile device can be uploaded to the Real-Time Transformation Interface server where they can be combined with AppStats collected by the Real-Time Transformation Interface server and analyzed by the customer's data analytics tools.

Statistics are captured first to a memory buffer, then flushed when specified by the application developer to a file in flash storage. This file is used for the datastore to hold application statistics during the lifecycle of the application. When the user wishes to export the data for further processing, the `Export` method is called, which flushes the remainder of the in-memory data, and then formats the data either into a relational database SQL dump for off-loading from the phone, or a JSON export format suitable for uploading to theReal-Time Transformation Interface.

By calling `stopRecord` and `startRecord`, the application can pause and resume recording statistics as often as desired during the session.

The AppStatistics singleton object is part of the utilities component of the SDK, and must be initialized with the `initAppStats` method prior to first use. This will ensure the datastore is initialized with the correct schema to hold all of the information to be gathered. Statistics will not be recorded until the application calls the `startRecord` method.

Note The `upgradeSchema` method has been deprecated for SDK 3.1.

If the schema changes after an update of an application that uses a newer version of Mobile SDK, then `initAppSats` will now silently delete and recreate the database, as the normal use case of App Statistics does not require all data from every device to be exported an uploaded.

General requirements for how to use app stats

1. Initialize App Stats

   For iOS
   Put the following code somewhere in application startup. A good place might be in the `AppDelegate "didFinishLaunchingWithOptions"` handler:

```
NSString * deviceID = [[[UIDevice currentDevice] identifierForVendor] UUIDString];
kfxKUTAppStatistics *appStatsObj = [kfxKUTAppStatistics appStatisticsInstance];
NSString *appStatsDataBasePath = [NSString stringWithFormat:@"%@/
yourfilename.sqlite",
  [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask,
 YES)
  lastObject]];       // This can be any file, name and path are application
 defined
[appStatsObj initAppStats:appStatsDataBasePath];
```

For Android

In order to implement event listeners for the various AppStatistics events (thresholds reached, write file status, export status, etc), you may wish to define a class responsible for managing your application's interface with AppStatistics, for example:

```
}
And here is the code to initialize AppStatistics and set various thresholds and
 listeners:
AppStatistics mSDKAppStatObj = AppStatistics.getInstance();
  mSDKAppStatObj.setDeviceId(UUID.randomUUID().toString());

 mSDKAppStatObj.setFileSizeThreshold(Integer.valueOf(ConstValues.sAppStatsFileThreshold)
 * 1000);

 mSDKAppStatObj.setRamSizeThreshold(Integer.valueOf(ConstValues.sAppStatsRAMThreshold)
 *1000);
  mSDKAppStatObj.addAppStatsExportListener(this);
  mSDKAppStatObj.addAppThresholdListener(this);
  mSDKAppStatObj.addAppStatsWriteFileListener(this);
String databasePath = "yourfilename.sqlite"; //This can be any file, name is
 application defined
        try{
            mSDKAppStatObj.initAppStats(databasePath);
        }
        catch(RuntimeException ex){

        }
```

2. Start recording

   For iOS

   ```
   [appStatsObj startRecord];
   ```

   For Android

   ```
   mSDKAppStatObj.startRecord();
   ```

3. Use SessionEvents as desired to log application specific custom events. If you are using Kofax Analytics for Mobile server side, see chapter 6 of the Kofax Analytics for Mobile Administrator Guide for details.
4. WriteToFile periodically

   App Statistics are recorded to an internal memory buffer. Periodically this should be written out to the database with the writeToFile call in the App Stats API.

The decision for when the application calls writeToFile is up to the application developer, but we recommend the following:

Try to writeToFile periodically so as not to fill up too much memory. The RAM size threshold event can be used to monitor this.

It is preferable to writeToFile while the application is relatively inactive, such as not in the middle of taking a picture or image processing. Sometimes an application navigation event such as returning to the main menu can be a sensible trigger point.

If there is a way to catch lifecycle events, try to write out the buffers before an application exits or is suspended to the background.

Note that you must stop recording state while writeToFile is called, then re-enable recording when it is complete. To do that, you will need to listen for the writeToFile complete event.

The code to writeToFile is as follows:

For iOS

```
if([self.appStatsObj isRecording]) {
                [self.appStatsObj stopRecord];
                [appStatsObj writeToFile];
}
```

And in the `writeFileStatus` delegate handler:

```
- (void) writeFileStatusEvent : (KUTappStatsWriteFile) type andProgress :
  (int) percentComplete withError: (int) errorCode withMsg: (NSString *) errorMsg{
   if(type==KUT_WRITEFILE_STATUS_COMPLETE)
{
// Re-enable recording
[self.appStatsObj startRecord];
}
}
```

For Android

```
if (mSDKAppStatObj.isRecording()) {
try {
                 mSDKAppStatObj.stopRecord();
                 mSDKAppStatObj.writeToFile();
}
catch (KmcRuntimeException ex) {
Log.e(TAG, "Exception in writeToFile: " + ex.getMessage());
}
            }
```

And in the writeFileStatus delegate handler:

```
 @Override
    public void writeFileStatusEvent(AppStatsWritetoFileEvent event) {


        if (event.getWritetoFileStatus() == WriteFileStatus.WRITE_STATUS_COMPLETE)
 {


                mSDKAppStatObj.startRecord();
```

```
 }
    }
```

5. Export the data to JSON periodically, for upload to Real-Time Transformation Interface.

   For iOS

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
 NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *exportPath;
    exportPath = [NSString stringWithFormat: @"%@/BR_Export.json",
 documentsDirectory];
    errorValue= [appStatsObj export:exportPath withFormat:KUT_EXPORT_TYPE_JSON];
[appStatsObj startRecord];
```

   For Android

```
  try {
            if (mSDKAppStatObj != null && mSDKAppStatObj.isRecording()) {
                canStartRecord = false;
                mSDKAppStatObj.stopRecord();
                mSDKAppStatObj.export(exportFilePath,
 ConstValues.sAppStatisticsExportFormat);
            }
        } catch (KmcRuntimeException ex) {
            Log.e(TAG, "Exception at manualExport() and Message :" +
 ex.getMessage());
        }
```

6. Upload Exported files to Real-Time Transformation Interface.

   In order to upload the exported JSON to Real-Time Transformation Interface for Kofax Analytics for Mobile to be able to aggregate and report on them, you will need the Real-Time Transformation Interface layer on a KTM server. There is a web service API at the following path:

   `http://<serverurl>/mobilesdk/api/appStats`

   Upload the contents of the exported JSON data using an HTTP PUT.

Recording App Statistics sessions

The App Statistics methods `beginSession`, `endSession`, and `logSessionEvent`, provide a means for an application to group application statistics that belong to the same application-defined session. Each session is a grouping in which all subsequent AppStats operations will be logged with the same `sessionKey` until the next `endSession` call.

The application begins by calling `beginSession`, passing in an application-defined `sessionKey` and category, each of which are text strings. This same `sessionKey` should also be passed to any server-side calls, such as the Real-Time Transformation Interface, so stats collected on the server can reflect the same `sessionKey`. `Category` is an application-defined string, representing the type of session.

Examples of typical categories include CheckDeposit, BillPay, etc, but it is completely up to the application to define a meaningful string. This is what will appear, upon export, in the Category field of the `SessionEvent` database table.

Check deposit example

For example, a check deposit session could consist of the following steps, all of which are considered part of the same "check deposit session":

1. `BeginSession`.
2. Capture a picture of the check (front) and a picture of the check (back) with Image Capture Control, which returns SDK Image objects.
3. Process the front Image object using SDK Image Processor to read the MICR on the front.
4. Process the back Image object using the SDK Image Processor to confirm the presence of an endorsement signature.
5. Submit the front check image to a Real-Time Transformation Interface Transformation Web service for extraction.
6. Present the extracted check fields to the user and allow corrections of fields that did not extract correctly.
7. Submit the corrected fields to a Real-Time Transformation Interface Validation Web service to validate the changed fields.
8. If validation passes, the check is electronically deposited via the application's secure proprietary server interface.
9. `EndSession`.

From an AppStats point of view, what ties all of these individual steps together is a unique, application-specified `SessionKey`. The `logSessionEvent` method accepts an `AppStatsSessionEvent` object as a parameter, which exposes a `SessionKey` property. The application can decide which steps contain valuable AppStats metrics, and log its type, event time, and result tied to a particular `SessionKey`.

In order to associate specific images, capture events, and image processing events with a specific `SessionKey`, `Document` and `Image` AppStats events and their corresponding database tables are included in the AppStats schema.

The Document objects contain Pages, and Pages contain Images. Since, from an AppStats perspective, a Page contains a single Image, there is no added value to logging Page creation events. Therefore, only Image creation events are logged.

Consequently, the database schema shows Images in the AppStats Image table directly linked to the AppStats Document table. The Document `AppStats` event and database table include a `SessionKey` field, which provides the link to one or more rows in the `SessionEvent` database table which have the same `SessionKey`. The entry in the `SessionEvent` database table was inserted or updated by a call to the `logSessionEvent` method on the AppStatistics object.

The `ImageProcessor` AppStats events and database table allow the collection of Image Processing metrics including start time, stop time, image processing profile, source image, processed image, and image processing results.

Related AppStats events

Using the same check deposit example, the table below shows AppStats events that would be recorded based on the application taking advantage of the `logSessionEvent` method and the `Document`, `Image`, and `ImageProcessor` AppStats events:

Steps and Events

Process Step	Associated Events
The application calls `beginSession`, passing in a unique `sessionKey`, and a Category of CheckDeposit.	
The application captures the front and back check images using the Image Capture control.	An AppStats `CaptureEvent` is logged which includes a link to the captured Image object in the Image table. The `Source` field of the Image database table row indicates the Image came from the Image Capture control.
The application sends the front side Image object to the SDK ImageProcessor object for processing to detect and return the MICR line. Crop and deskew are also performed.	
The SDK ImageProcessor object processes the image, and creates a new processed Image object.	An AppStats event is logged in the Image database table indicating the `CreationTime` of the Image, and the `FileSize`, if its `Image Representation` property indicates a file-based representation. The `Source` field in the AppStats Image table indicates the Image was created by the ImageProcessor.
	An AppStats event is logged in the `ImageProcessorEvent` database table indicating processing start time, stop time, the EVRS Op String that requested MICR recognition, the EVRS result code, and links to these Image rows in the AppStats Image table: 1. Original unprocessed Image. 2. Processed Image. The `ProcessingProfile` field in the `ImageProcessorEvent` database table indicates that MICR recognition was requested.
The application decides it wants to document the results of the MICR reading step as the first in a sequence of steps required to complete an entire check deposit session.	The application calls the `logSessionEvent` method and passes in a new instance of the `AppStatsSessionEvent` data-only Class. One of the properties of this object is a unique `SessionKey` created by the application which corresponds to a single check deposit session. The `Type` property of the `AppStatsSessionEvent` object is set to `ReadMICR`.
The application sends the back side Image object to the SDK ImageProcessor object for processing to detect the existence of a signature on the endorsement line.	

Process Step	Associated Events
The SDK ImageProcessor object processes the image through EVRS, and creates a new processed Image object.	An AppStats event is logged in the Image database table indicating the `CreationTime` of the image, and the `FileSize` (if its `Image Representation` property indicates a file-based representation). The `Source` field in the Image table indicates the image was created by the ImageProcessor. The `ProcessingProfile` field indicates that handprint recognition was requested.
	An AppStats event is logged in the `ImageProcessorEvent` database table indicating processing start time, stop time, the EVRS Op String that requested handprint recognition, the EVRS result code, and links to these Image rows in the AppStats Image table: 1. Original unprocessed Image. 2. Processed Image . The `ProcessingProfile` field in the `ImageProcessorEvent` database table indicates that handprint detection was requested.
The application decides it wants to document the results of the endorsement signature reading step as the next in a sequence of steps required to complete an entire check deposit session.	The application calls the `logSessionEvent` method and passes in a new instance of the `AppStatsSessionEvent` data-only class. One of the properties of this object is the same application-created `SessionKey` as above which corresponds to a single check deposit session. The `Type` property of the `AppStatsSessionEvent` object is set to `VerifyEndorsement`.
Assuming the `ReadMICR` and `VerifyEndorsement` steps are successful, the application is ready to submit the captured, processed front-side check image to the Real-Time Transformation Interface Transformation Web service for extraction.	The application calls the `logSessionEvent` method and passes in a new instance of the `AppStatsSessionEvent` data-only class. One of the properties of this object is the same `SessionKey` previously used by the application to log `ReadMICR` and `VerifyEndorsement` steps. The `Type` property of this `AppStatsSessionEvent` object is set to `RTTI_CheckExtraction_Start`. The `EventTime` field of the AppStats `SessionEvent` table is set to the current time by the `logSessionEvent` method.

Process Step	Associated Events
The Real-Time Transformation Interface Transformation Web service is called. Included in the HTTP Request is: 1. `SessionKey`. 2. Cropped, deskewed front-side check JPEG, TIFF, or PNG image.	The Real-Time Transformation Interface Transformation Web service response includes the `SessionKey` sent in the HTTP Request. The application calls the `logSessionEvent` method and passes in a new instance of the `AppStatsSessionEvent` data-only class, and then it calls `logSessionEvent` to add a new row to the `SessionEvent` table. The `Type` property of this `AppStatsSessionEvent` object is set to `RTTI_CheckExtraction_End`. The `EventTime` field in the `SessionEvent` table row indicates when the Real-Time Transformation Interface response was received. The `Response` field in the `SessionEvent` table row is also updated by the application to indicate Real-Time Transformation Interface Transformation success or failure (the application defines the contents of this field).
	The Real-Time Transformation Interface response also includes classification information indicating how many documents were created from the images sent in the Real-Time Transformation Interface request. Each document in the response shows a `DocumentID` (which must be a GUID). So, from the response, it's clear to the application which images belong to the same document.
The application creates a new SDK Document object.	The object logs an AppStats `Document` event in the Document database table. The application sets the `DocumentID` property of the SDK Document object to the same `DocumentID` returned by Real-Time Transformation Interface. This new row in the Document database table will have a primary key ID matching the `DocumentID` in the response from Real-Time Transformation Interface. The `SessionKey` property of the SDK Document object is set to the same `SessionKey` used in earlier calls to the `logSessionEvent` method. The `SessionKey` field of the AppStats Document table is updated to match this.
The application creates a new Page object and adds it to the Document object created in the previous step.	This does not generate an AppStats `Page` event, as `Page` creation is not a relevant metric; only the image creation event is. Based on the Real-Time Transformation Interface response, the application knows which images belong to a given document.
The application adds the front-side check Image object to the Page object created in the previous step by calling the `addImage` method.	This causes an AppStats Image table update operation to be performed, which updates the `DocumentID` field in the Image table to link with its parent AppStats `Document` table row.

Process Step	Associated Events
	The `DocumentID` property of the `Image` event database row is now set to the parent `Document` event row in the `Document` AppStats table. This is a critical link, as it joins the AppStats `Document` events generated on the Real-Time Transformation Interface server with the mobile SDK AppStats `Image` events. AppStats analytics tools on the server can discover information about the original captured image(s), and what EVRS processing has been performed on those images based on just the Document ID.
The fields of the Document object are updated by the application, with the extracted field results from the Real-Time Transformation Interface Transformation response.	The application calls the `getFields` method on the Document object containing the image that was submitted for extraction. Each field value is updated by calling the `setValue` method on the Field object.
The values of extracted fields are presented by the application to the user and the user makes corrections as needed.	An AppStats `FieldChangeEvent` is recorded for every field that gets changed by the user. The `FieldChangeEvent` database table is linked to the Document table through the `DocumentID` field. **Note** This is a breaking change from earlier versions, where the `FieldChangeEvent` database table was linked to the `Environment` table.
The application calls the `logSessionEvent` method and passes in a new instance of the `AppStatsSessionEvent` data-only class.	One of the properties of this object is the same `SessionKey` previously used by the application to log the `ReadMICR` and `VerifyEndorsement` steps. The `Type` property of this `AppStatsSessionEvent` object is set to `RTTI_CheckValidation`. The `EventTime` field of the `SessionEvent` table is set to the current time by the `logSessionEvent` method.
The Real-Time Transformation Interface Validation Web service is called. Included in the HTTP Request is: 1. `SessionKey`. 2. Field name-value pairs, including fields corrected by the user.	The Real-Time Transformation Interface Validation Web service response includes the `SessionKey` sent in the HTTP Request. The application calls `logSessionEvent` to add a new row, setting the `EventTime` field in the `SessionEvent` table row to indicate when the Real-Time Transformation Interface response was received. The `Response` field in the `SessionEvent` table row is also updated by the application to indicate Real-Time Transformation Interface Validation success or failure (application defines the contents of this field).
If Validation fails for one or more fields, the `IsValid` and `ErrorDescription` properties of the SDK Field object are updated by the application. The user is asked to make corrections to conform with validation requirements.	An AppStats event is generated to record: • Original field value. • Corrected field valu.e • `IsValid` flag • ErrorDescription text indicating cause of validation failure. This event is recorded in the `FieldChangeEvent` table.

Process Step	Associated Events
The previous two Process Steps are repeated until check validation succeeds for all fields.	
The application calls `endSession`. The entire check deposit session is complete. All associated AppStats events on both the mobile device and the Real-Time Transformation Interface server which track the entire sequence have been logged and connected together through: 1. A single `sessionKey`. 2. One or more `DocumentID` values.	

SQL database schema

App Statistics are recorded internally in an SQLite datastore of the following schema (see figure below). This same data schema is used by the `Export` method as a SQL dump that can be imported to a server side database for further analysis.

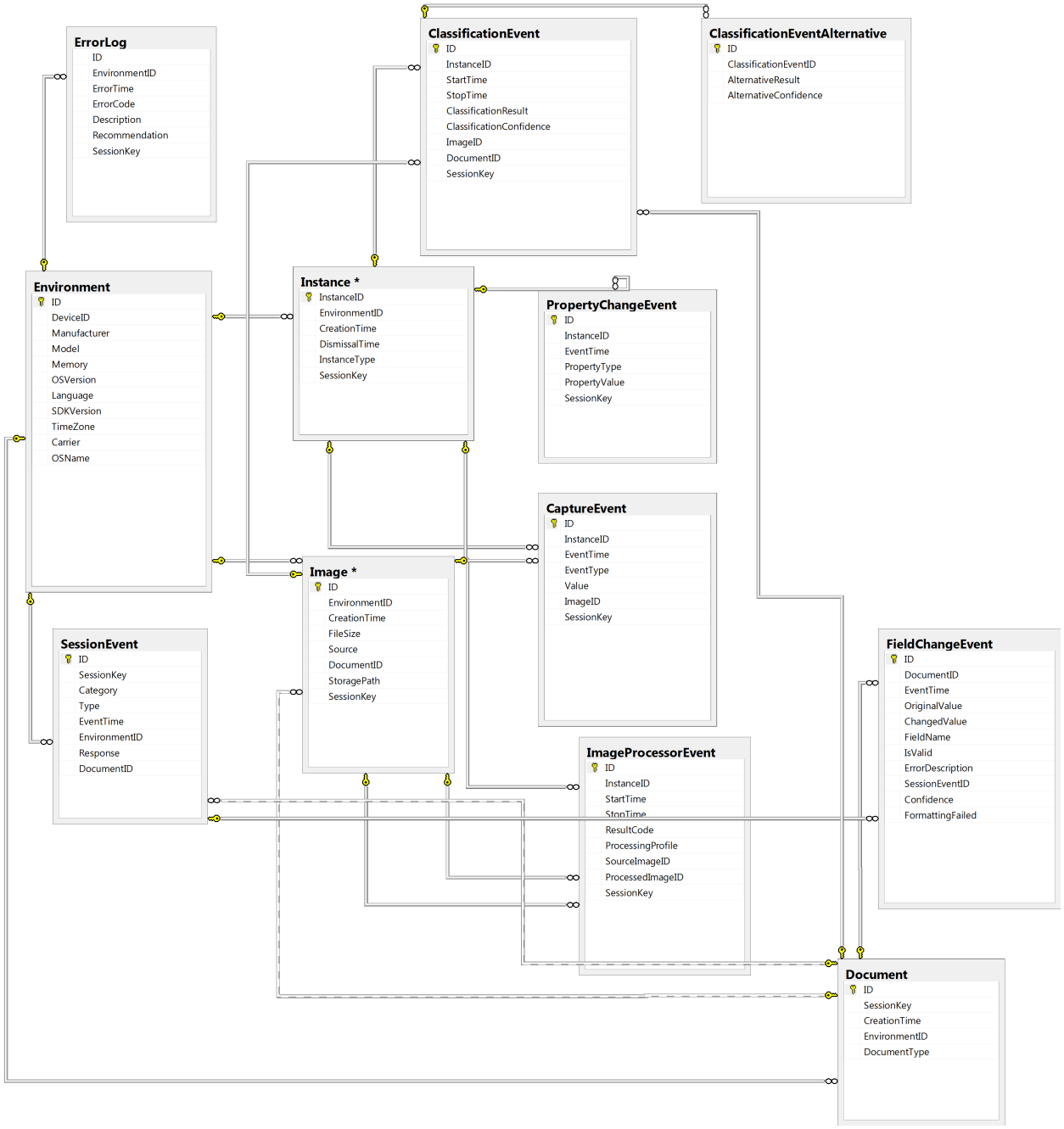
Note Statistics are recorded in the order of occurrence. No additional formatting or filtering capabilities are provided.

The App Statistics are organized by events, some of which belong to an instance of an SDK object. For example, an instance of an image capture control may trigger multiple image capture events.

So in this case, when the image capture control is instantiated, the SDK will create a record in the Instance table, with an `InstanceType` corresponding to the Image Capture control. Subsequently, each time an image is captured with the capture control, a record is created in the CaptureEvent table, with the `InstanceID` of the instance control that is being used.

All time data are in the form of UTC-based date/time strings.

Note For convenience in importing and aggregating data from multiple devices into a central database, all foreign keys are GUIDs (Globally Unique Identifiers), as opposed to auto-increment integers.



CaptureEvent table

The events fired while capturing an image are stored in this table. For example, `PageDetect`, level indicator events, etc.

Database Column	Data Type	Description
ID	GUID	Primary key for this table.
InstanceID	GUID	Foreign key for the Instance table.
EventTime	DateTime	Time event occurred.
EventType	Text	String showing the type of capture event. • Stability • Levelness • Focus • PageDetect • Capture • ForceCapture • CaptureExperienceCapture
Value	Integer	Numeric value associated with the event type.
ImageID	GUID	Foreign key to Image table.
SessionKey	Text	Key that identifies the active session.

Document table

Each time an SDK Document object is created, an entry is created in this table.

Database Column	Data Type	Description
ID	GUID	Primary key for this table.
SessionKey	Text	Key that identifies the active session.
CreationTime	DateTime	Time the document was created.
EnvironmentID	GUID	Foreign key to the environment table.

Environment table

Information about the device and SDK version is stored in this table.

Database Column	Data Type	Description
ID	GUID	Primary key for this table.
DeviceID	Text	GUID Identifier of the device used as assigned by the application.
Manufacturer	Text	Name of device manufacturer.
Model	Text	Model of the device used. For iOS, the model text follows official Apple nomenclature.
Memory	Text	Amount of memory in the device used.
OSVersion	Text	Name and version of OS.
Language	Text	Language setting of OS.

Database Column	Data Type	Description
SDKVersion	Text	Version of SDK.
TimeZone	Text	Time zone setting for the device.
Carrier	Text	Name of Phone Carrier.
OSName	Text	Name of OS (iOS, Android).

ErrorLog table

If there are errors, like crashes, the error will be logged in this table. If the error code returned is as expected, it will be logged in an individual event table.

Database Column	Data Type	Description
ID	GUID	Primary key for this table.
EnvironmentID	GUID	Link to relevant row in Environment table.
ErrorTime	DateTime	Date and time of error.
ErrorCode	Integer	Code for reported error.
Description	Text	Textual description of error code.
Recommendation	Text	Error cause, if any.
SessionKey	Text	Key that identifies the active session.

FieldChangeEvent table

The purpose of this table is to record the field and data corrections the user had to make to the data returned by the server. For example, the Real-Time Transformation Interface returns DL (Driver License) information, if the user changes the first name to match the DL, the change and field changed will be recorded in this table. A query to this table will help the fields that are not getting processed correctly.

Database Column	Data Type	Description
ID	GUID	Primary key for this table.
DocumentID	GUID	Foreign key to Document table.
EventTime	DateTime	Time event occurred.
OriginalValue	Text	Value prior to change.
ChangedValue	Text	Value after change.
FieldName	Text	Name of changed field.
IsValid	Integer	1 = new value found, else 0.
ErrorDescription	Text	Description of any errors related to this changed field, such as field validation results.
SessionEventID	GUID	Optional, foreign key to SessionEvent table.
Confidence	Float	Confidence value ranging from -1 to 1.
Formatting Failed	Bool	Indicates that formatting failed.

Image table

Each time an Image object is created by the SDK, either from image capture, creating from a file, or as a result of an image processing operation, a new entry will be created in the Image table.

Database Column	Data Type	Description
ID	GUID	Primary key for this table.
EnvironmentID	GUID	Foreign key to the Environment table.
CreationTime	DateTime	Time the instance was created.
FileSize	Integer	Size of file (if image is file-based). If image is not file-based, this field will be 0 until the image is written out to a file.
Source	Text	Source of the image (Image capture, user created, Image processing, etc.)
DocumentID	GUID	Foreign key to Document table. If this image is not part of a document (or more technically, not part of a page of a document), this field can be null.
StoragePath	Text	If the application chooses to store a copy of this image somewhere, this field can be set to record where that image is persisted. This can be done on the server-side Real-Time Transformation Interface App Stats plugin.
SessionKey	Text	Key that identifies the active session.

ImageProcessorEvent table

This table records all image processing events, and tracks the source, processed images, time it took to process, and the processing profile that was used.

Database Column	Data Type	Description
ID	GUID	Primary key for this table.
InstanceID	GUID	Foreign key to the Instance table.
StartTime	DateTime	Time the event started.
StopTime	DateTime	Time the event stopped.
ResultCode	Integer	Result of image processing operation.
ProcessingProfile	Text	Image processing profile used (human-readable string describing settings in a `BasicSettings` profile, or the actual formatted processing string for an image perfection profile).
SourceImageID	GUID	Foreign key to Image table.
ProcessedImageID	GUID	Foreign key to Image table.
SessionKey	Text	Key that identifies the active session.

Instance table

Each time an AppStats-aware SDK object is instantiated, a row is inserted in this table. The table contains `instanceId` (a GUID) which is its `PrimaryKey` (PK) and corresponds to a foreign key (FK) for the other tables. Hence, the relationship between the Instance table and an event table is a one-to-many relationship.

If there are two objects of the same type in the current application, the creation time and dismissal time of each object are recorded in a separate row in the table.

Database Column	Data Type	Description
InstanceID	GUID	Primary key for this table.
EnvironmentID	GUID	Foreign key to the Environment table.
CreationTime	DateTime	Time the instance was created.
DismissalTime	DateTime	Time the instance was dismissed.
InstanceType	Text	Type of instance: • ImageClassifier • ImageProcessor • BarcodeReader • ImageCapture • ImageReviewEdit • BarcodeCapture • FrontOfficeServer • TotalAgilityServer
SessionKey	Text	Key that identifies the active session.

PropertyChangeEvent table

If the user has to make changes in settings, such as roll, pitch and so forth, in order to capture a picture, it's recorded in this table. Also if the user is unable to take a picture and has to force a manual image capture, those settings are also captured in this table.

Database Column	Data Type	Description
ID	GUID	Primary key for this table.
InstanceID	GUID	Foreign key to the Instance table.
EventTime	DateTime	Time event occurred.
PropertyType	Text	String showing which property changed: • Stability Delay • Levelness Threshold Pitch • Levelness Threshold Roll • Page Detect Mode • Continuous Capture • GPS Usage

Database Column	Data Type	Description
PropertyValue	Text	New value for the property that changed.
SessionKey	Text	Key that identifies the active session.

Session Event table

The `SessionEvent` table, along with the corresponding new method `LogSessionEvent`, provides a mechanism for an application to log time stamps in App Statistics. An example of a complete session depositing a check, where the beginning may be when an image is captured on the phone, carrying it through being processed, sent to the server for extraction, receiving the response back, and finally creating a Document object based on extracted fields.

As many entries in the `SessionEvent` table can be created as the application developer chooses, but each would use the same `SessionKey`, and the free-text "Type" field would describe the type of event (example: Image processed, server request, response received, etc). This allows the application developer to record key points of interest for timing measurements, which could then be used in subsequent analysis for performance measurements of various parts of the process.

Database Column	Data Type	Description
ID	GUID	Primary key for this table.
SessionKey	Text	Key that identifies the active session.
Category	Text	Maps to the current project (bill pay, check deposit, etc.).
Type	Text	Application-defined description of session event type.
EventTime	DateTime	Time of the event.
EnvironmentID	GUID	Foreign key to the environment table.
Response	Text	Optional. Application-provided response from a particular session event. For example, a server response.
DocumentID	GUID	Optional, foreign key to document table.

Exporting data

The SDK supports exporting data from the datastore to MSSQL. When the export is done, an SQL dump file will be created at the location specified by the user in the `Export` method. For convenience, there is a sample create script in the AppStats folder of the product zip file that creates an MSSQL database that matches the supported schema.

The developer can create their own custom exporter or use the export capability provided with the SDK. With custom exporters you can export data from the datastore to a database of you choice.

With the SDK's built in exporter, the data from datastore will be exported to an SQL dump file as a series of insert statements in MSSQL syntax. No create statements are included in this export file. It assumes the target database already exists.

If an SQL dump file already exists, the SDK will throw an exception. If you want to preserve multiple SQL dumps, the application will have to create a new file.

SDK-provided export

The export handler provided in the SDK exports the data from the datastore to an SQL dump file in the location specified in the `Export` method.

In order to use the exporter in the SDK, you have to add the export listener in the application and the event provided with the listener will return the status of the export.

JSON export format

The SDK supports exporting data from the datastore to a database-neutral format. The format is simple JSON and is suitable for sending to the Real-Time Transformation Interface via a new Real-Time Transformation Interface Web service designed to accept AppStats data. When the export is done, a JSON file with an extension of ."json" will have been created at the location specified by the user in the `Export` method.

JSON-formatted export data is structured as a single JSON object which contains two name/value pairs:

- `AppStatisticsVersion`: The `AppStatisticsVersion` contains the version of the SDK which generated this JSON data.
- `AppStatisticsTables`: JSON object which contains a name/value pair for each database table. The name is the table name, the value is an array of objects (one for each table row). Each object in the array contains a name/value pair for every field in that table row, with the name of each field paired with its value.

A sample implementation for an Android device is as follows, however the basic format will be the same regardless of platform:

```
{ "AppStatisticsVersion": "2.2.0.0.0.0",
  "AppStatisticsTables" :
 {
  "Environments" :
   [ {"ID": "ab6cb9d6-2e13-4067-9ed0-ecf453bd7a46",
    "DeviceID": "",
    "Manufacturer": "Samsung",
    "Model": "Samsung Galaxy Nexus",
    "Memory": "711480 kB",
    "OSVersion": "4.0.4",
    "Language": "en_US",
    "SDKVersion": "2.0.0.0",
    "TimeZone": "PDT",
    "Carrier": "Verizon",
    "OSName": "Android"
     }
   ],
     "Instances" :
      [ {"InstanceID": "024f2227-c6a7-4ad3-bcf1-cb98fb562397",
         "EnvironmentID": "ab6cb9d6-2e13-4067-9ed0-ecf453bd7a46",
         "CreationTime": "2014-05-29 15:54:57",
         "DismissalTime": "",
         "InstanceType": "ImageClassifier"
        }
      ],
     "ClassificationEvents":
```

```
      [ {"ID": "c636d6bf-0b22-475c-b740-a0c582b3b39e",
         "InstanceID": "024f2227-c6a7-4ad3-bcf1-cb98fb562397",
         "StartTime": "2014-05-29 15:55:16",
         "StopTime": "2014-05-29 15:55:16",
         "ClassificationResult": "LA1",
         "ClassificationConfidence": "0.8337612"
        }
      ],
     "ClassificationEventAlternatives":
      [ {"ID": "833acf7b-c8c5-4393-abc4-e21a60cdb374",
         "ClassificationEventID": "c636d6bf-0b22-475c-b740-a0c582b3b39e",
         "AlternativeResults": "MS1",
         "AlternativeConfidence": "-0.36527917"
        },
        {"ID": "60398A01-85B0-44d4-95D1-A818380EE016",
         "ClassificationEventID": "c636d6bf-0b22-475c-b740-a0c582b3b39e",
         "AlternativeResults": "CA2",
         "AlternativeConfidence": "-0.69517412"
        }
      ],
     "FieldChangeEvents" :
      [ {"ID": "798251f3-0208-4863-bab8-8982752fc702",
         "DocumentId: : "a963aa90-9718-449a-ab5b-5efd6b1e0021",
         "EventTime" : "2014-07-30 16:26:36",
         "OriginalValue" : "dog",
         "ChangedValue" : "cat",
         "FieldName" : "Pet_Name",
         "IsValid" : "1",
         "ErrorDescription" : ""
        }
      ]
    }
}
```

Custom exporter (Android)

You can create your own custom exporter if you have a need to modify the standard exported output that comes with App Statistics.

1. Create a class that implements `AppStatsDsExportHandler`.
2. Register the handler using the `Register` method provided in the SDK.
3. Add the export listener and pass in the correct format type.
4. The `AppStatsDsExportHandler` returns the `AppStatsDaoField` array and table name. The rows that were retrieved from the SQLite database are returned in the `AppStatsDaoField` array. This array has to be parsed into `INSERT` statements acceptable to the database of choice.

The sample implementation is as follows:

```
public class DatabaseExportHandler implements AppStatsDsExportHandler {

 private static final String TAG = "Handler";
 String path;
 ArrayList<String> insertStatements = new ArrayList<String>();

 public DatabaseExportHandler() {

 }
```

```
@Override
public void onExportAppStatsRowEvent(String dsExportTblName,
  AppStatsDaoField[] daoFields) {

 writeInsertStatementsToDumpFile(this.path, dsExportTblName, daoFields);
}

@Override
public void configDsExpFilePath(String dsExpFilePath) {
 path = dsExpFilePath;

 if (path.endsWith(".sql"))
  createDumpFile(this.path);
 else {
  this.path = this.path + ".sql";
  createDumpFile(this.path);
 }
}

/**
 * This method writes INSERT statements for the rows of data
 * returned by the handler
 * into the sql dump file specified by the user in the export method.
 *
 * @param path - export file path with .sql as the extension
 * @param tableName - table into which the insert statements will be applied
 * @param rowInformation - rows returned in the handler
 */
private void writeInsertStatementsToDumpFile(String path, String tableName,
  AppStatsDaoField[] rowInformation) {
 File myFile = new File(path);
 String insertStatment = null;
 StringBuffer insertStNames = new StringBuffer();
 StringBuffer insertStValues = new StringBuffer();
 insertStNames.append("INSERT INTO " + tableName + " (");
 insertStValues.append("VALUES (");
 if (myFile.exists()) {
  for (AppStatsDaoField r : rowInformation) {

   insertStNames.append(r.getDsFieldName() + ",");
   if (r.getDsFieldType() == AppStatsDsFieldType.DS_FIELD_TYPE_FLOAT)
   {
    insertStValues.append("'" + r.getDsValueFloat() + "'"
      + ", ");
   }
   else if (r.getDsFieldType() == AppStatsDsFieldType.DS_FIELD_TYPE_INTEGER)
   {
    insertStValues.append("'" + r.getDsValueInt() + "'" + ", ");
   }
   else if (r.getDsFieldType() == AppStatsDsFieldType.DS_FIELD_TYPE_LONG)
   {
    insertStValues.append("'"+ (r.getDsValueLong()) + "'"+ ", ");
   }
   else if (r.getDbFieldType() == AppStatsDbFieldType.DB_FIELD_TYPE_DATETIME)
   {

    if(r.getDsValueString() == "0" || StringUtils.isEmpty(r.getDsValueString())
    || r.getDsValueString() == null)
    {
     Log.i(TAG,"value[0] = "+r.getDsValueString());
     insertStValues.append("'"+ (" ") + "'"+ ", ");
    }
    else
    {
```

```
     Log.i(TAG,"value[0 - oops] = "+r.getDsValueString());
     insertStValues.append("'"+ (r.getDsValueString()) + "'"+ ", ");
    }
   }
   else if (r.getDsFieldType() == AppStatsDsFieldType.DS_FIELD_TYPE_STRING) {
    if (r.getDsFieldName().contains("ID")) {
     String guid = r.getDsValueString();
     insertStValues.append("'" + "{" + guid + "}" + "'"
       + ", ");
    } else
    {
     //remove extra ' - bug
     String value = r.getDsValueString().replace("'", "");
     insertStValues.append("'" + value + "'"
       + ", ");
    }
   }
  }

  insertStValues.deleteCharAt(insertStValues.lastIndexOf(","));
  insertStNames.deleteCharAt(insertStNames.lastIndexOf(","));
  insertStNames.append(")");
  insertStValues.append(");");
  insertStatment = insertStNames.toString()
    + insertStValues.toString();
  Log.i(TAG, "insertStatment = " + insertStatment);
  insertStatements.add(insertStatment);
 }
 try {
  FileOutputStream fOut = new FileOutputStream(myFile);
  OutputStreamWriter myOutWriter = new OutputStreamWriter(fOut);
  for (int i = 0; i < insertStatements.size(); i++) {
   myOutWriter.write(insertStatements.get(i) + "\n");

  }

  myOutWriter.close();
  fOut.close();
 } catch (Exception e) {
  e.printStackTrace();
 }
}

/**
 * This method creates the sql dump file on the sdcard
 * @param path - export dump file with .sql extension appended if absent
 */
private void createDumpFile(String path) {

 if (path == null || StringUtils.isEmpty(path))
  throw new NullPointerException();

 int lastIndex = path.lastIndexOf(File.separator);
 String parentPath = path.substring(0, lastIndex);
 String fileName = path.substring(lastIndex, path.length());
 File newFolder = null;
 if (!parentPath.equalsIgnoreCase(Environment
   .getExternalStorageDirectory().getPath())) {
  newFolder = new File(parentPath);
  if (!newFolder.exists()) {
   newFolder.mkdir();
  }

  File sqlDumpFile = new File(newFolder, fileName);
```

```
   try {
    sqlDumpFile.createNewFile();
   } catch (IOException e) {
    e.printStackTrace();
   }
  } else {
   File file = new File(parentPath, fileName);
   try {
    file.createNewFile();
   } catch (IOException e) {
    e.printStackTrace();
   }
  }

 }
}
```

Custom exporter (iOS)

You can create your own custom exporter if you have a need to modify the standard exported output that comes with App Statistics.

You can do this simply by using an option delegate method:

```
-(NSString *) formatExportRow:(NSString *) tableName withFields:
        (NSMutableArray *) daoFields;
```

If you implement this method in your delegate, it will be called. If not, then you will get the default export format you have now. An example of an implementation for your delegate method, that produces the same results as the current built-in export (be sure to `#import "kfxKUTAppStatsDaoField.h"`):

```
-(NSString *) formatExportRow:(NSString *) tableName withFields:
         (NSMutableArray *) daoFields
{
    NSMutableString * fieldnames = [NSMutableString string];
    NSMutableString * fieldvalues = [NSMutableString string];

    unsigned long count = [daoFields count];

    for (int i = 0; i < count; i++)
    {
        kfxKUTAppStatsDaoField * field = (kfxKUTAppStatsDaoField *)
        [daoFields objectAtIndex:i];

        [fieldnames appendString:field.fieldName];
        if (field.fieldType == KUT_APP_STATS_DAO_FIELDTYPE_GUID)
        {
            [fieldvalues appendString:[NSString stringWithFormat:@"'{%@}'",
            field.fieldValue]];
        }
        else
        {
            [fieldvalues appendString:[NSString stringWithFormat:@"'%@'",
            field.fieldValue]];
        }

        if (i < count - 1)
        {
            [fieldnames appendString:@","];
            [fieldvalues appendString:@","];

        }
```

```
    }

    NSString * result = [NSString stringWithFormat:@"INSERT into %@ (%@)
    values (%@);\r\n", tableName, fieldnames, fieldvalues];

    return result;
}
```

About hybrid apps using PhoneGap

PhoneGap is an open source mobile application development framework, based upon the Apache Cordova project. See `cordova.apache.org` for documentation for details. The PhoneGap Mobile Plugin for the mobile SDK in your mobile application can be used to capture and process images and bar code data received from mobile devices.

For more information see the *PhoneGap Plugin Developer's Guide* at `\Hybrid\PhoneGap \Documentation\KofaxPhoneGapdevelopersguide_EN.pdf` in the installation folder.

Note
The earliest Cordova version supported by the Kofax PhoneGap Plugin is version 5.0. The latest Cordova version supported by the Kofax PhoneGap Plugin is version 6.3

Kofax mobile plugin for Kofax TotalAgility

The PhoneGap based Kofax Mobile Plugin for TotalAgility makes it possible to access mobile and tablet forms in Kofax TotalAgility, which utilize the new Mobile Capture and Mobile Bar Code Capture controls. By using this plugin in your mobile application, you can use your application to capture and process images and bar code data received from mobile devices.

For more information see the *PhoneGap Plugin Developer's Guide* at `\Hybrid\PhoneGap \Documentation\KofaxPhoneGapdevelopersguide_EN.pdf` in the installation folder.

Serialization and deserialization

The Logistics and Engines data classes have a serialization feature that allows certain objects instantiated from these classes to be written out to a specified file, and later restored from this file. The purpose of this feature is:

- To conserve memory space
- To save certain capture data for later use
- To be able to restore objects from the archived files
- To store certain objects for later use when a document is created. This reduces server interaction and improves performance.

In addition:

- **Compatibility**: The library supports backward compatibility, such that an application that uses a newer version of the SDK can still restore objects from archives created by an older version of the SDK library. However, it is not a design goal to archive and unarchive with forward compatibility, such as decoding an archive on a newer version of the class, with an older version of the library.
- **Versions**: Archiving a particular object includes the version of the parent library to which it is associated. This allows for verifying backward compatibility before use. The library throws an exception if the deserialization process cannot be performed with an input file.
- **Named serialization**: Each archive includes the name of the class so that the library can detect errors with improper usage while maintaining backward compatibility. The library throws an exception when the archive file does not match the class association. Therefore, each serialization file includes two permanent items, namely, the version of the library and the class name.

Serializable classes

The SDK logistics and engine libraries are designed to allow archiving certain classes. These classes contain the unique data associated with documents and images. These are the only classes that can be serialized. Any of these classes may be serialized from any starting point, or from any level in a hierarchy of objects, and individually. But, the most common use case is to save work in progress associated with documents.

The following logistics and engine data classes can be serialized. Serializing a parent object serializes all child objects when those objects are valid. There are a maximum of 5 levels of hierarchy, and the capability ensures all lower-level objects are serialized correctly.

Logistics Classes:

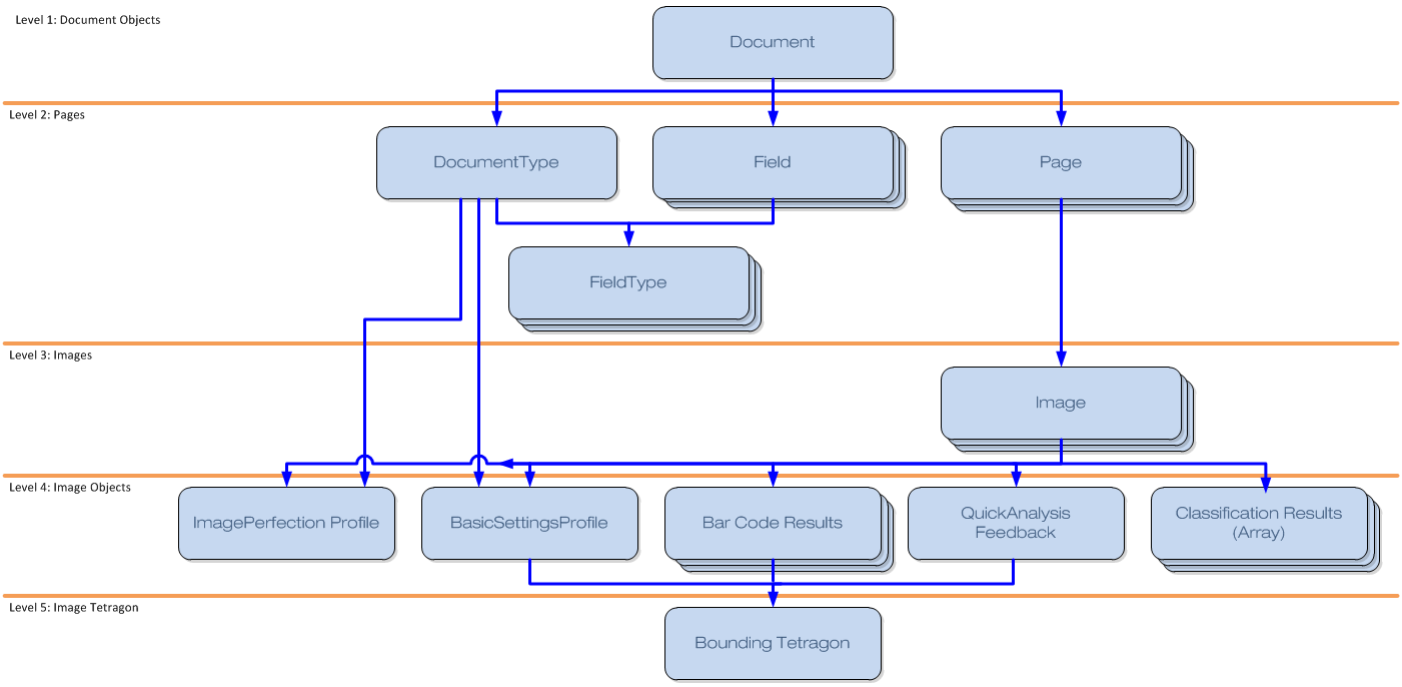
- `Document`
- `DocumentType`
- `Field`
- `FieldType`
- `Page`
- `Image`

Engine Data Classes

- `ImagePerfectionProfile`
- `BasicSettingsProfile`
- `QuickAnalysisFeedback`
- `BarcodeResult` (part of an array in Image)
- `BoundingTetragon`
- `ClassificationResult` (part of an array in Image)

Serialization hierarchy

The diagram below shows the serialization hierarchy and dependencies.



Serialization Hierarchy

The SDK library includes classes organized under a top level Document object. If a document is serialized, every Logistics and Engine Data object shown in the figure is included in the archive. In some cases, the serialization file will include an array of objects or a set of objects. The "…" in the diagram indicates a one-to-many relationship in the hierarchy. For instance, a document may be composed of several pages, and a page may have multiple images within it. An image may have an image perfection profile or a basic settings profile used by image processing. It may also have a bar code results array, a classification results array or some quick analysis feedback results data. All of these Engine Data objects would be included in the output file.

In the largest case, a document would include all five levels of objects in the hierarchy. A basic settings profile may include a bounding tetragon, while the quick analysis feedback object may refer to a different bounding tetragon and a bar code results object may refer to yet another.

Serialization of images

The Image object includes an image represented in different formats. These formats include: bitmap, a buffered file, a stored file or none at all. The table shows the various representations using the enumerations in the Image class. These are the only valid combinations, and the serialization feature saves images in all these combinations.

Image Representation	File Representation	Description
Unknown	Unknown	An object with no associated image
Bitmap	n/a	An object that contains a native bitmap image: either a source image or a processed image.

Image Representation	File Representation	Description
File	Stored	An object that has a path to a file that contains the image stored in a file (as represented by a file path with a named file that contains the image saved in a specific mime type).
File	Buffered	An object that contains an embedded internal buffer representing the image, which is normally smaller than the bitmap image. The buffer contains the image formatted according to a specified mime type, such as JPG or TIFF.
Both	Stored	An object that has both an embedded bitmap and a file path with a file name that contains the image saved in a particular mime type.
Both	Buffered	An object that has both an embedded bitmap and a file image buffer that contains the image formatted in a particular mime type.

Conditions and limitations

The application that uses the SDK objects is responsible for initiating, saving, and restoring data. SDK classes that can be serialized implement the needed methods for both serializing and deserializing.

The application may use serialization for any purpose that it deems appropriate. The application layer decides the naming convention to use for the archive files, but we recommend using, at least, the name of the top level root class within the file name to help manage the correct association.

The application manages saving the archive file names in some way that allows restoration. The library does not maintain any archival file names in any class within the SDK.

An application can only restore an object from an archive file associated with the object to which it was intended, or the library throws an exception.

An application can only restore an object from an archive file that was written by the current library version or an earlier one. Otherwise, the library throws an exception.

Android specifics

Objects

The following objects are serialized:

- KfxEngines
  - BarCodeResult
  - BasicSettingsProfile
  - BoundingTetragon
  - Image
  - ImagePerfectionProfile
  - QuickAnalysisFeedback

    - ImageClassificationResult
- KfxLogistics
    - Document
    - DocumentType
    - Field
    - Image
    - FieldType
    - Page
    - WscIndexField

Example: application serialization

The following sample serializes the `ImagePerfectionProfile` object:

```
ImagePerfectionProfile myIPP = new ImagePerfectionProfile();
  File dest = new File(Environment.getExternalStorageDirectory() +
 java.io.File.separator
 + "sdk20IPP.save");

  myIPP.setIpOperations("_Do90DegreeRotation_4_DoBinarization_
_DoSkewCorrectionAlt__DoScaleImageToDPI_200");
  myIPP.setName("Insurance Sample Case");
  myIPP.setIpOperationsFilePath(dest.getPath());

  try {
   // write the object to file
   FileOutputStream fos = new FileOutputStream(dest);
   ObjectOutputStream out = new ObjectOutputStream(fos);
   out.writeObject(myIPP);
   out.close();
  } catch (KmcRuntimeException e) {
   e.printStackTrace();
  } catch (Exception ex) {
   ex.printStackTrace();
  }
```

Example: application deserialization

The following sample deserializes the `ImagePerfectionProfile` object:

```
try {
   // read the object from file
   FileInputStream fis = new FileInputStream(dest);
   ObjectInputStream in = new ObjectInputStream(fis);
   ImagePerfectionProfile myIPPCopy = (ImagePerfectionProfile)in.readObject();
   in.close();
  } catch (KmcRuntimeException e) {
   e.printStackTrace();
  } catch (Exception ex) {
   ex.printStackTrace();
  }
```

iOS Specifics

The serialization capability adopts the `NSCoding` protocol so that the object may be archived and unarchived using the iOS standard keyed archival methodology.

The `NSCoding` delegate methods use encoded key value pairs so that the decode method can read keyed values from the archive in any order, while ignoring new keys it does not know about.

The application uses the following approach to archive an object or a hierarchy of objects.

```
[NSKeyedArchiver archiveRootObject:<object> toFile: archiveFileName];
```

Using keyed archive files, you can save this object for later use. To restore an object or a hierarchy of objects from a named archive file, the application calls the `NSKeyedUnarchiver` method, using the following approach.

```
myRestoredObject = [NSKeyedUnarchiver
unarchiveObjectWithFile:archiveFileName];
```

When you initialize an object with the `NSKeyedUnarchiver`, iOS restores objects in a similar way to an implied alloc-init. The library only throws exceptions when it detects something wrong with the deserialization operation.

The libraries use unique key names that have the class name as the prefix, so that all keys are guaranteed to be unique to the Kofax class names and properties. The `encodeWithCoder` method encodes the object, and encodes each property using the unique key name. iOS handles saving that encoding to the file specified with the `KeyedArchive` method. When the application calls the `unarchiveObjectWithFile`, it uses the previously saved file by name.

Example: application serialization

This shows how to archive a bounding tetragon that is first initialized with test data.

```
// Example of serialization for only this pre-initialized object we
will use as the root object.

 kfxKEDBoundingTetragon * myTetragon = [[kfxKEDBoundingTetragon alloc] init];

 CGPoint aPoint;

 // Setup an object for example purposes
 aPoint.x = 77.500000;
 aPoint.y = 66.250000;
 myTetragon.topLeft = aPoint;

 aPoint.x = 55.120000;
 aPoint.y = 44.500000;
 myTetragon.topRight = aPoint;

 aPoint.x = 33.500000;
 aPoint.y = 22.750000;
 myTetragon.bottomLeft = aPoint;

 aPoint.x = 11.12500;
 aPoint.y = 0.500000;
 myTetragon.bottomRight = aPoint;
```

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString* documentsDirectory = [paths objectAtIndex:0];
NSString * archiveFileName = [documentsDirectory stringByAppendingPathComponent:
@"SerialTetragonObject.ar"];
[NSKeyedArchiver archiveRootObject:myTetragon toFile:archiveFileName];
```

Example: application deserialization

All keyed unarchive operations must be wrapped in a try-catch block because the `initWithCoder` method throws exceptions. This example shows a popup with the error description when the library throws an exception on unarchive.

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString* documentsDirectory = [paths objectAtIndex:0];
NSString * archiveFileName = [documentsDirectory stringByAppendingPathComponent:
@"SerialTetragonObject.ar"];

kfxKEDBoundingTetragon * myTetragonRestored;
kfxKEDImage * myImageRestored;

int errorCount = 0;

 @try{
     myTetragonRestored = [NSKeyedUnarchiver unarchiveObjectWithFile:archiveFileName];
 }@catch (NSException *anException)
 {
     dispatch_async(dispatch_get_main_queue(), ^{
         NSString * strException = [NSString stringWithFormat:@"Unarchive Exception:
         %@",anException.name];

         UIAlertView* alert = [[UIAlertView alloc] initWithTitle:strException message:
         anException.reason delegate:self cancelButtonTitle:@"OK"
otherButtonTitles:nil];
         [alert show];
     });
     errorCount++;
 }
```

Licensing

All apps using the Kofax Mobile SDK require a license.

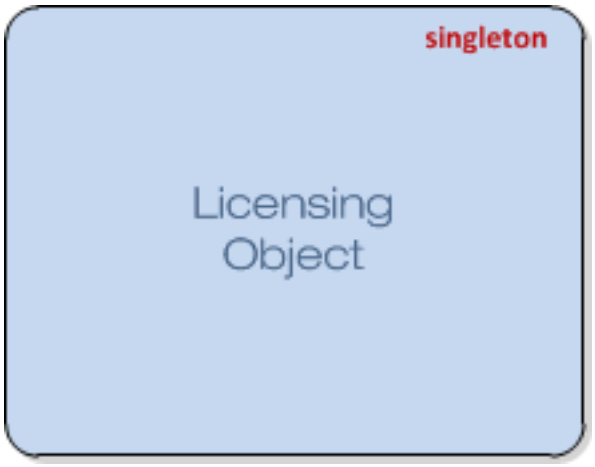
The license string is supplied as a text string to include in your software project. For iOS, the license file can be directly included in the project. For Android, the license string needs to be copied into the Java code.

In addition to the methods below, you can also use a bar code reader to capture the SDK license at run time. For more information, see License capture control object.

Licensing object

Can be found in Utilities.

Licensing is an object that comprises the licensing mechanism for this SDK. The licensing mechanism prevents unlicensed users from using key components of the mobile SDK.



Licensing Object Diagram

The following mobile SDK objects are protected by licensing checks:

- `ImageCaptureControl`
- `BarCodeCaptureControl`
- `ImageReviewEditControl`
- `ImageProcessor`
- `BarCodeReader`
- `ImageObject`
- `AppStatsClass`
- `FrontOfficeServer`
- `ImageClassifier`

Licensing basics

It is necessary to obtain a license in order to use the protected SDK objects. This string is supplied as a file that can be included in a software project. To set the license, the application calls the `setMobileSDKLicense` method of the License object. The method returns `KMC_SUCCESS` if the license is valid for use. The method returns an error code if the license has expired or the license string is otherwise invalid.

The application can call `getDaysRemaining` to determine when the license expires, which returns a value for how many days are left in the license. The `setMobileSDKLicense` method returns a non-zero error code if it was invalid, in which case the return value for `getDaysRemaining` is zero.

kfxEVRS_License.h

The file `kfxEVRS_License.h` contains a non-working license string to ensure that you can at least compile the sample source, if not actually run the sample program. You will need to obtain a working license, according to the terms of your contract.

A sample file is shown here.

Sample: kfxEVRS_License.h

```
//============================================================================
// kfxEVRS_License.h
//
// Copyright (c) 2018 Kofax. All rights reserved. Kofax Confidential.
// Unauthorized use, duplication, distribution, or disclosure is strictly
// prohibited.
// Kofax VRS Mobile SDK
//
// This license header file contains a placeholder license string for use
// in the example code to ensure that you can compile the sample source.
// You may obtain an evaluation license from Kofax for the purpose of
// evaluating the Kofax VRS Mobile SDKlibrary.
// You will receive a production license from Kofax, according to the terms of
// your contract.
//
//
//============================================================================

#ifndef kfxEVRS_License_h
#define kfxEVRS_License_h

#define PROCESS_PAGE_SDK_LICENSE      "This is not a valid license."

#endif
```

Driver license classifier

Note ImageClassifier has been deprecated in Mobile SDK 3.4.0.

Can be found in Engines.

The API includes an image classifier for United States Driver Licenses. The SDK includes some of the specific configuration and model xml files to configure the classifier to classify U.S. Driver Licenses. There can only be one instance of the classifier at a time.

Once you load the configuration file and the model file, you can use the classify Image method, which examines an input image and returns an array of classification result data as a list of possible classifications, with the most confident classification returned first in the array. The classification data is stored in the input kfxKEDImage object. One classification result indicates what document type is associated with the image.

An application can use the classification data `classID` string to associate it with a particular DocumentType object in the application. For example, "CA2" may indicate a type 2 California driver license.

Before classifying a driver license image

In the sample application included with the SDK, an image processing string, similar to the following example, is used for classifying driver licenses. Note that this string is provided here for example purposes only. You should refer to the source code for the application to obtain the most current version of this string.

```
_DeviceType_2_DoSkewCorrectionPage__DoCropCorrection_
_DoScaleImageToDPI_500_DocDimSmall_2.125_DocDimLarge_3.375_LoadSetting_
<Property Name=\"CSkewDetect.prorate_error_sum_thr_bkg_brightness.Bool\"
Value=\"1\" Comment=\"DEFAULT 0\"></Property>_LoadSetting_
<Property Name=\"CSkwCor.Do_Fast_Rotation.Bool\" Value=\"0\"
Comment=\"DEFAULT 1\"></Property>"
```

Note The above string (or a similar one) must be used to process the image before attempting to classify it.

Initialize the classifier

In order to use the classifier, the application must first initialize it. To do so, specify the model file and the configuration file. Two sample files are provided in the SDK to initialize the Classifier for detecting US Driver Licenses. These files must be bundled with the application. The SDK includes the XML configuration files in the `Configuration Files` folder. Copy these files into your application project, and provide a fully qualified path to these files when you initialize the classifier.

It is the app's responsibility to download the latest classifier files from the extraction server to update the files on the mobile device. To do this, use Web services calls to the extraction server where the server administrator has stored the appropriate classifier files.

Note At this time, the model can be updated only by the products team.

Classification results

Classification results are stored in the `ClassificationResult` objects array in the input KEDImage object. To determine the highest confidence result, use the first `ClassificationResult` in the objects array. This object includes the orientation of the image in 90 degree increments, and the confidence value.

The application can use the classification result data to perform advanced image processing using specific settings for that document type. For instance, one sheet may have a dark background, while another has a light background.

Each possible classification result object includes a class ID, the orientation of the image string, and a score value. The score is a float, with negative values indicating low probability, while positive values indicate high confidence. The higher the positive score, the more confident the classification.

On-Device Extraction

On-device extraction affords the app developer the choice of extracting data from an image of an ID on the device instead of sending large images to a server for extraction. Extraction is done entirely on the device; however, the component does periodically communicate with a Real-Time Transformation Interface or TotalAgility server for license accounting purposes.

The on-device extraction API is a simple interface that expects a region, front and/or back images of the ID, and a callback/delegate. It is possible to only include the regions you will require in your app, and leave any of the others out. In case region files do not exist for the region selected for extraction, the engine will throw an exception through the returning interface.

For specific details regarding ID fields, please refer to the *Kofax Mobile ID Capture Administrator's Guide*.

Using On-Device Extraction

When using on-device extraction, it is necessary to provide the extraction engine with some asset files. These assets can either be pre-loaded into your applications assets (iOS and Android), pre-loaded onto a device's external storage (Android only), or loaded from the Kofax Quick Updater for Mobile download link (iOS and Android).

Each option has its own implementation, each located in the `com.kofax.mobile.sdk.extract.id` package. By default, the SDK will use the `LocalProjectProvider`.

Because the extractor will use the `LocalProjectProvider` by default, you can simply use the default constructor with a context parameter. For example:

Android example

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        IIdExtractor extractor = new OnDeviceIdExtractor(this);
        extractor.extractFields(...);
    }
}
```

iOS example

```
kfxKOEIDExtractor* idExtractor = [kfxKOEIDExtractor new];
idExtractor.delegate = self;
[idExtractor extract:...];
```

When using the `LocalProjectProvider` it is necessary to copy the project zip to the local storage location of the device prior to running any on-device extraction. See the *Kofax Mobile SDK API Reference Guide* of the `LocalProjectProvider` for more details.

If you would like to use the `ServerProjectProvider`, ensure you have a Kofax Quick Updater for Mobile instance running, and provide an instance of the `ServerProjectProvider` to the `OnDeviceIdExtractor`. For example:

Android example

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        IProjectProvider projectProvider = new ServerProjectProvider(this, "http://
myCompanyServer.com/odedldservice/api/odedownload/");
        IIdExtractor extractor = new OnDeviceIdExtractor(this, projectProvider);
        extractor.extractFields(...);
        ...
    }
}
```

iOS example

```
KFXServerProjectProvider* serverProvider = [[KFXServerProjectProvider alloc]
 initWithURL:[NSURL URLWithString:@"http://myCompanyServer.com/odedldservice/api/
odedownload/"]];
kfxKOEIDExtractor* idExtractor = [[kfxKOEIDExtractor alloc]
 initWithProjectProvider:serverProvider];
idExtractor.delegate = self;
[idExtractor extract:...];
```

Be sure to provide the entire base path to the Kofax Quick Updater for Mobile web API.

It is also possible to use your own implementation of `IProjectProvider` to allow for complete control of communication between your application and the updater instance. Simply implement the `IProjectProvider` interface, and supply your implementation to the `OnDeviceIdExtractor`, similar to the examples above. Please see the *Kofax Mobile SDK API Reference Guide* for `IProjectProvider` for more information.

Furthermore, it is possible to use your own implementation of the cache provider, controlling the behavior of the reference implementations of `IProjectProvider`. Simply implement the `IBundleCacheProvider` interface, and supply your implementation to the `IProjectProvider` instance of your choice. Provide that `IProjectProvider` to the `OnDeviceIdExtractor`, similar to the examples above. Please see the *Kofax Mobile SDK API Reference Guide* for `IBundleCacheProvider` for more information.

Project providers use cache providers for caching extraction assets. There is a built-in cache provider - `BundleCacheProvider`. It is possible to customize the directory where cached assets will be stored. In order to do so, create an `IBundleCacheProvider` instance with specified directory path and supply it to the project provider instance of your choice. For example:

Android example

```
IBundleCacheProvider cache = new BundleCacheProvider(new File("/path/to/cache"));
IProjectProvider projectProvider = new ServerProjectProvider(MyActivity.this, cache,
    "http://myCompanyServer.com/odedlservice/api/odedownload");
IIdExtractor extractor = new OnDeviceIdExtractor(MyActivity.this, projectProvider);
```

iOS example

```
KFXBundleCacheProvider* cache = [[KFXBundleCacheProvider alloc] initWithPath:@"/path/
to/cache"];
KFXServerProjectProvider* serverProvider =
  [[KFXServerProjectProvider alloc] initWithURL:
  [NSURL URLWithString:@"http://myCompanyServer.com/odedldservice/api/odedownload/"]
  cacheProvider:cache];
```

```
  kfxKOEIDExtractor* idExtractor = [[kfxKOEIDExtractor alloc]
  initWithProjectProvider:serverProvider];
idExtractor.delegate = self;
[idExtractor extract:...];
".
```

Downloading package updates

In order to enable incremental updates to On-Device Extraction, the client-server architecture allows the downloading of extraction configuration and model files.

The client is able to download the project packages (`fields.xml`, `classifier.config/.model`, and `cities.zip`) in order to complete classification without downloading the entire region's variants. Once classification is complete and the variant is known, the client can download just that variant package to complete extraction. The client can also request a bulk download. In this case, selecting a region will download the region package as well as all of the variants onto the device.

Android example for project package downloads

```
try {
    final String[] projectVersion = new String[1];
    IProjectProvider projectProvider = new ServerProjectProvider(mainActivity, "http://
myCompanyServer.com/odedldservice/api/odedownload/");
    ICompletionListener listener = new ICompletionListener() {
        @Override
        public void onComplete(Object o, Exception e) {
            projectVersion[0] = (String) o;
        }
    };
    projectProvider.getHighestVersion("USIDs", SdkVersion.getSdkVersion(), listener);
    ...
    // download project packages
    projectProvider.getProject("USIDs", projectVersion[0], listener);
} catch (MalformedURLException e) {
    Log.e("ServerProjectProvider", "MalformedURLException");
}
```

iOS example for project package downloads

```
__block NSString* wversion;
KFXServerProjectProvider* serverProvider = [[KFXServerProjectProvider
 alloc]initWithURL:[NSURL URLWithString:@"http://myCompanyServer.com/odedldservice/api/
odedownload/"
]];
[serverProvider getHighestVersion:@"USIDs" sdkVersion:[[kfxKUTSdkVersion sdkInstance]
 getSdkVersion] completionHandler:^(NSString *version, NSError *error) {
        wversion = version;
    }];
[serverProvider getProject:@"USIDs" version: wversion completionHandler:^(NSString
 *path, NSError* error) {
    }];
```

Android example for variant downloads

```
projectProvider.getVariant("USIDs", "Variant_Classified", projectVersion[0], listener);
```

iOS example for variant downloads

```
[serverProvider getVariant:@"Variant_Classified" forProject:@"USIDs" version:wversion
 completionHandler:^(NSString *path, NSError* error) {
                NSLog(@"Variant load finished error: %@", err);
```

```
 }];
```

Android example for bulk downloads

```
projectProvider.loadAllVariantsForProject("USIDs", listener);
```

iOS example for bulk downloads

```
[serverProvider loadAllVariantsForProject:@"USIDs" completionHandler:^(NSError* error){
            NSLog(@"loadAllVariantsForProject error: %@", error.description);
            });
```

ID extraction licensing

The app can use the `acquireVolumeLicenses` method in the SDK to pre-allocate on-device extraction license volume units obtained from the server. This form of preallocation is useful, for example, for situations where the device cannot connect to the server.

Outstanding license usage will be reported to the license server when the server connection is available, and either acquireVolumeLicenses or setMobileLicenseServer is called.

However, once the licenses have been downloaded to the device, there is no way to restore them to the license server. Also, please keep in mind that these preallocated, on-device extraction volume licenses are permanently decremented on the license server. Any unused volume remaining on a device will be lost if the application is uninstalled or if the application data is otherwise cleared.

For ID extraction, use the following code to set the licensing.

Android

```
LicensingVolume.setMobileSDKLicenseServer("http://my-server-here/mobilesdk",
 Licensing.LicenseServerType.RTTI);
Licensing.setCertificateValidatorListener(new CertificateValidatorListener() {

    @Override
    public SSLSocketFactory getSSLSocketFactory(String s) {
        Log.i("Licensing", "validation challenge");
        return null;
    }
});

Licensing.addVolumeLicenseEventListener(new Licensing.VolumeLicenseEventListener() {
    @Override
    public void licenseOperationFailed(Licensing.VolumeLicenseFailureData
 volumeLicenseFailureData) {
        Log.e("Exception", "License Operation Failed");
    }

    @Override
    public void licenseOperationSucceeded(Licensing.VolumeLicenseResultData
 volumeLicenseResultData) {
        Log.i("Licensing", "License Operation Succeeded");
    }
});
Licensing.acquireVolumeLicenses(Licensing.LicenseType.ID_EXTRACTION, numberOfLicences);
```

iOS

```
kfxKUTLicensing* _license = [[kfxKUTLicensing alloc] init];
    _license.delegate = self;
    _license.certificateValidatorDelegate = self;
```

```
   [_license setMobileSDKLicenseServer:@"http://my-server-here/mobilesdk"
type:SERVER_TYPE_RTTI];

   self.expectation = [self expectationWithDescription:@"License"];
   [_license acquireVolumeLicenses:LIC_ON_DEVICE_EXTRACTION
withCount:numberOfLicences];


   - (void)acquireVolumeLicenseDone:(int) licAcquired error: (NSError*) error
   {
       NSLog(@"acquireVolumeLicenseDone: count: %d error: %@", licAcquired,
error.description);
   }
```

Optical Character Recognition engines

On-device extraction uses several Optical Character Recognition (OCR) engines. The Tesseract OCR engine is available for non-Latin character recognition (such as Arabic, Cyrillic, and Greek). For iOS, the Tesseract OCR libraries are already included in the SDK. For Android, the following libraries are required by the Tesseract OCR engine:

- tesseract-4.0.0.5.jar
- liblept.so
- libtess.so

If you do not require the extraction of non-Latin characters, these Tesseract OCR libraries can be removed safely. No errors will be thrown.

For iOS, the Tesseract OCR engine is disabled by default so that its binaries do not increase application size. However, an error is returned if the application attempts to extract non-Latin text. If your application reads non-Latin languages, enable the Tesseract OCR engine by doing the following:

1. Call the `enable KFXTesseractOCR` class method.
2. Run `kfxKOEIDExtractor` to extract data with the Tesseract OCR engine.

Diagnostics and error codes

You can enable the display of diagnostics information during the capture experience. This feature provides state information about status of automatic capture and help with debugging. This view is supported for the Document, Check, FixedAspectRatio and Passport capture experiences. Logging is turned off by default.

Following diagnostics information is displaying as an overlay over the capture experience:

- Focus state
- Stability value and thresholds
- Selected camera resolution
- Pitch/roll - Value and thresholds
- Level value based on pitch/roll
- Detected page
- Focus Area

- Pre-crop target frame

Note The Focus Area feature is applicable and enabled only for the Android platform for these specific devices with poor focus: Samsung Galaxy S5, Samsung Note 3, Samsung S4 mini, HTC One max, SONY Xperia Z, LG G3, Moto G, Lenovo Yoga, Nexus 7, Nexus 9, Asus Zenfone, HTC One M9.

Enabling diagnostics in the capture experience

Methods to manage logging information for the capture experience UI are provided for the Android and iOS platforms. Below are examples showing how to enable diagnostics:

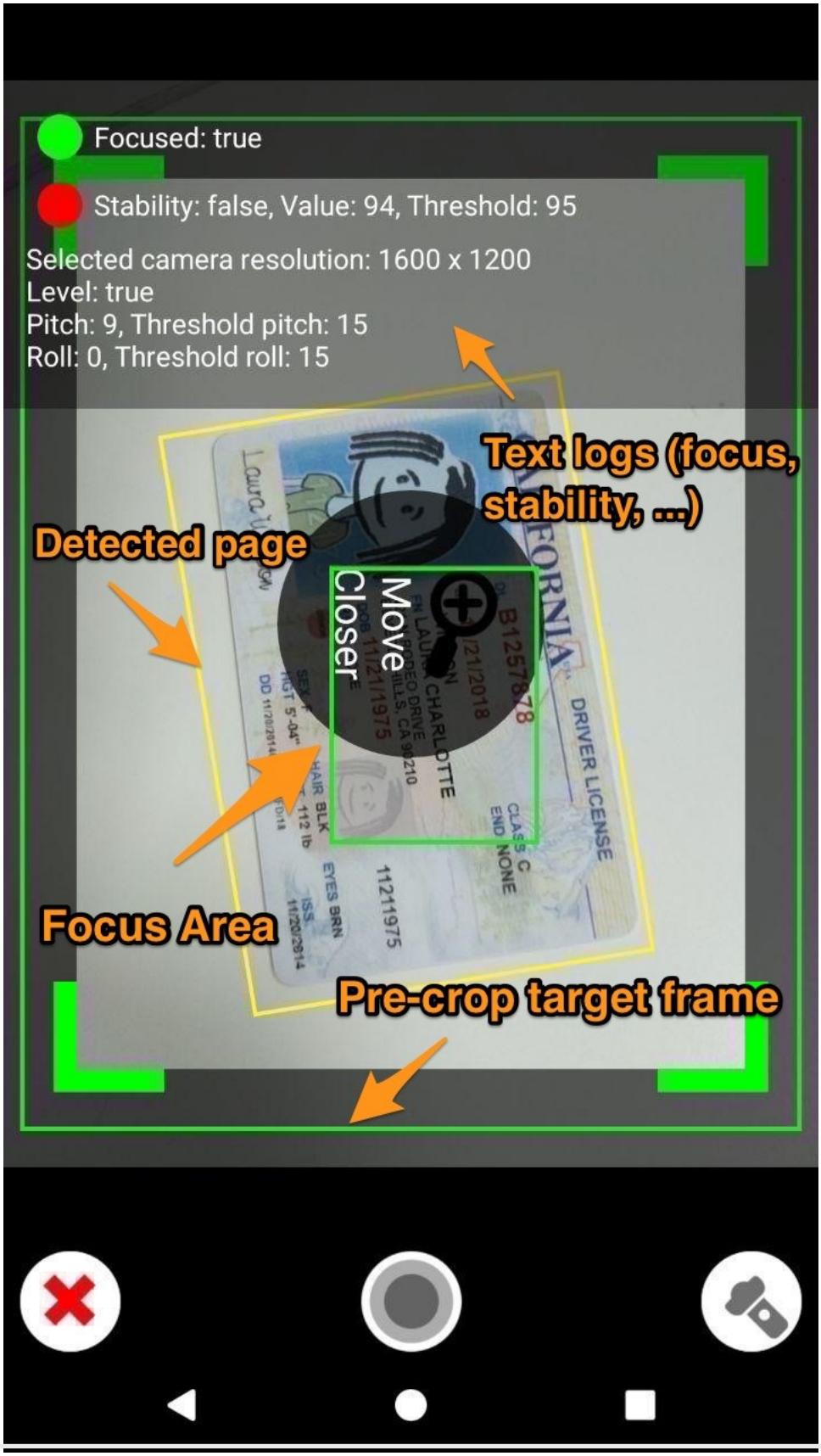
Android

```
DocumentBaseCaptureExperience captureExperience
captureExperience.setDiagnosticsViewEnabled(true);
```

iOS

```
kfxKUIDocumentBaseCaptureExperience* captureExperience;
captureExperience.diagnosticsViewEnabled = TRUE;
```

This example screen shows the debugging user interface.


Focused: true
Stability: false, Value: 94, Threshold: 95
Selected camera resolution: 1600 x 1200
Level: true
Pitch: 9, Threshold pitch: 15
Roll: 0, Threshold roll: 15
Text logs (focus, stability, ...)
Detected page
Move Closer
Focus Area
Pre-crop target frame

Error code strings

For a complete listing of error codes, refer to the following files.

Note For best results, view the files with a text editor optimized for development use.

Strings for the SDK error codes are available within the release zip separately for each platform:

Android

The SDK error message strings are available as XML files at the following path location within the release zip: Android\MobileSDK_libs\jar\localization.

iOS

The SDK error message strings are available in the bundle file at the following path location within the release zip: iOS\Frameworks\MobileSDK.zip\SDKStrings.bundle

About the Kofax mobile demo application

The Kofax Mobile Demo sample application is included with the SDK and may be found at `iOS\SampleApps\Native\KofaxMobileDemo-iOS.zip Android\SampleApps\Native\KofaxMobileDemo-Android.zip`. It demonstrates the capabilities of the mobile SDK. This demo is not intended to be used directly in a production environment. Rather, it is intended to demonstrate how the SDK can be used to support typical use cases. The Kofax Mobile Demo source code is packaged with the SDK and can be copied and used as a starting point for development.

Kofax Mobile Demo demonstrates SDK capabilities such as image capture, image processing, classification, and submitting images to KTM (Kofax Transformation Module) or KTA (Kofax TotalAgility).

Note When submitting bitonal images, it may be necessary to increase the maximum buffer size. If the buffer is too small, the application may present an error message stating that the maximum message size quota for incoming messages has been exceeded. The default value for this quota is 65K (65536) bytes. To increase the size, edit `maxBufferSize="65536"` in the Real-Time Transformation Interface `web.config` file.

It also demonstrates several use cases. These are briefly explained below.

Kofax server support

Kofax Mobile Demo can connect to the mobile frameworks (Bill Pay, Remote Check Deposit & Mobile ID) on the Kofax TotalAgility platform. Users can select the server type, Kofax Transformation Module (via the Real-Time Transformation Interface) or KofaxTotalAgility for each component in Kofax Mobile Demo. Some features that are currently supported on Real-Time Transformation Interface are not supported on the Kofax TotalAgility platform. Please see the comparison table below for more details on which features are supported.

Supported Parameters

Feature	Kofax TotalAgility	Real-Time Transformation Interface
Text Value	Y	Y
ErrorDescription	Y	Y
Confidence	Y	Y
Valid	Y	Y
CheckReasonReject	Y	Y
RestrictiveEndorsment	Y	Y
RestrictiveEndorsmentPresent	Y	Y
High light extracted data Coordinates	N	Y
formattingFailed	N	Y
fieldAlternatives	N	Y
Save original image	N	Y

Check deposit

The user begins by selecting a region and then capturing images for both sides of the check. After providing the amount on a check, the front and back images are captured. These captured images are submitted to servers which can handle the processing of the check.

Most similar applications just take images and submit them, however such images are typically large. Kofax Mobile Demo, on the other hand, processes the images by converting them to bitonal and by cropping to remove excess edge space. This is performed on the device and consequently reduces the size of image sent to the server.

Kofax Mobile Demo guides the user when taking photos of both the front and the back of a check. The mobile SDK, which is used to take the pictures of the checks, provides two modes: video and image. Kofax Mobile Demo uses the video mode for devices supporting high resolutions and image mode for devices with low resolution.

Once the image of a check front is captured, Kofax Mobile Demo performs the necessary image processing in the background. While this is happening, the user can proceed to take a picture of the check back. Once image processing for the front is complete, the image of check front is submitted to the server where check data is extracted and showed in a summary view. If there are issues, the user can retake the picture to correct them.

As part of the extraction process, a variety of validations are performed to test the results, thereby helping to ensure quality data. The server-side application checks if the image is too light or too dark. It also checks for various fields on the check. For example, on the front side of the check it looks for the payee name, amount, MICR code, and signature. Also, if a MICR line is not present, then Kofax Mobile Demo informs the user that the MICR was not found and user needs to verify if he has really submitted the front side of the check. If the same check is submitted again then the user is shown a duplicate check duplicate message.

On the back side, the signature (endorsement) is checked.

Once both the front and back images are submitted, the server extracts the data and responds with the information in the fields on the check. The check information and validation results are shown in detail in the "Check Information" screen. This screen provides not only the values obtained from processing the check, but also a confidence level for each value. There are also tabs for IQA Results, and Usability Results.

Note The SDK does not currently have a method for obtaining values for the confidence and valid fields from Kofax TotalAgility. Instead, Kofax Mobile Demo obtains these values by using Kofax Transformation Module to return the information via the Real-Time Transformation Interface.

The user may continue after reviewing these details. Clicking on "Make Deposit" submits the check to the bank. Please note that the check is not actually deposited since this is a demo application.

From the "Check Deposit" screen, the user can see a recent history of submitted checks. The history has the front and back image, check number, amount, MICR and date.

Check deposit settings

Check Deposit has several settings that the user can change.

- **Camera Settings** The user can choose from a variety of settings such as Gallery, Page Detection, Stability Delay and Roll Threshold.
- **Processing Settings** The user can configure various settings related to the processing of images. Parameters like AutoCrop, Auto Rotate, Deskew, Scale (dpi), and Sharpen can be configured. Deskew by content or layout can also be configured.
- **Server Settings** The user can configure the server to which the image will be submitted for extraction. The URL can be HTTP or HTTPS.
- **Advanced Settings** The user can enable MICR detection, or whether Check Validation and Check Extraction are performed locally or on the server. The user can also enable hand print recognition and duplicate check detection.
- **Edit Component Labels** The user can modify the text used for various field labels use on the screens.

Pay bills

Kofax Pay Bills, available from the main menu, allows the user to capture an image of a bill coupon and pay the bill. The user begins by selecting a region and then capturing images of the bills.

After capturing an image of the coupon, relevant data, such as the amount, due date, and payee information are extracted, or entered. After the required information is provided and confirmed, the user can tap "Make Payment" to pay the bill.

If, while taking the picture, the page is not detected, or the picture is not taken promptly, the user can opt to manually take the picture.

As with Check Deposit, the Pay Bills module also performs image processing on the device. The image processing parameters can be changed in the settings.

The processed image is sent to the server. The server extracts the values of the important fields and sends the results back to the application. After validating these values, the user can continue to add more payees.

"Make Payment" is similar to "Add Payee." Additional fields such as the amount are shown to the user.

Please note that the bill is not actually paid since this is a demo application.

Pay bills settings

Pay Bills has several settings that the user can change.

- **Camera settings** The user can choose from a variety of settings such as Gallery, Page Detection, Stability Delay and Roll Threshold.
- **Processing settings** The user can configure various settings related to the processing of images. Parameters like AutoCrop, Auto Rotate, Deskew, Scale (dpi), and Sharpen can be configured. Deskew by content or layout can also be configured.
- **Server settings** The user can configure the server to which the image will be submitted for extraction. The URL can be HTTP or HTTPS.
- **Edit component labels** The user can modify the text used for various field labels use on the screens.

ID card

The ID Card feature can be used to capture an ID card from a number of regions around the world. The user begins by selecting a region and then capturing images for both sides of the card. The captured images are then sent for data extraction.

Further more, ID verification is done by comparing the photograph on the ID against a selfie portrait captured using the Selfie Capture Experience in both Kofax TotalAgility and Real-Time Transformation Interface servers.

The information is shown to the user, which he can then verify and correct as needed and finally submit.

Please note that the card is not actually used since this is a demo application.

Note The ID Card feature requires a color image.

ID card settings

ID Card has several settings that the user can change.

- **Camera settings** The user can choose from a variety of settings such as Gallery, Page Detection and Stability Delay.
- **Processing settings** The user can configure various settings related to the processing of images. Parameters like AutoCrop, Auto Rotate, Deskew, Scale (dpi), and Sharpen can be configured. Deskew by content or layout can also be configured.
- **Server settings** The user can configure the server to which the image will be submitted for extraction. The URL can be HTTP or HTTPS.
- **Edit component labels** The user can modify the text used for various field labels use on the screens.

Credit Card

The credit card capture feature can be used to capture an embossed or a non-embossed credit card, which can then be used as a form of payment. The user begins by capturing an image of the card. The captured image is processed for data extraction.

Note Extraction from non-embossed cards is supported only on the server.

The information is shown to the user, which he can verify and correct as needed and finally submit.

Please note that the credit card is not actually used since this is a demo application.

Credit card settings

Credit Card has several settings that the user can change.

- **Extraction Settings** Credit card has support of Card IO as well as Server Extraction. By default Card IO is selected. If the server option is selected, the user can configure the server to specify which the image will be submitted for extraction. The URL can be HTTP or HTTPS. Also the below additional options will be enabled when server is selected
  - **Camera settings** The user can choose from a variety of settings such as Gallery, Page Detection and Stability Delay.
  - **Processing settings** The user can configure various settings related to the processing of images. Parameters like AutoCrop, Auto Rotate, Deskew, Scale (dpi), and Sharpen can be configured. Deskew by content or layout can also be configured.
- **Edit component labels** The user can modify the text used for various field labels use on the screens.

Passport

The passport feature can be used to capture the information page from a passport, which can then be used as a form of identification. The user begins by capturing an image of the information page. The captured image is sent for data extraction.

ID verification is performed by comparing the passport photo against a selfie portrait captured by the Selfie Capture Experience in both Kofax TotalAgility and Real-Time Transformation Interface servers.

We recommend using the Image mode for passports, since this mode produces images with a higher resolution. Using the Video mode is not recommended because it may negatively impact the extraction results.

The information is shown to the user, which he can verify, correct as needed and finally submit. Please note that when the Submit button is pressed in the Demo program, nothing happens. If On-Device Extraction is used, passport data is not actually sent anywhere. However, if server side extraction was used, data was sent to the server earlier in the workflow. However, none of this data is retained at the server.

Note The passport feature requires a color image.

Passport settings

Passport has several settings that the user can change.

- **Camera settings** The user can choose from a variety of settings such as Gallery, Page Detection, Stability Delay and Roll Threshold.
- **Processing settings** The user can configure various settings related to the processing of images. Parameters like AutoCrop, Auto Rotate, Deskew, Scale (dpi), and Sharpen can be configured. Deskew by content or layout can also be configured.
- **Server settings** The user can configure the server to which the image will be submitted for extraction. The URL can be HTTP or HTTPS.
- **Edit component labels** The user can modify the text used for various field labels use on the screens.

Custom Component

The custom component feature can be used to capture the information page from a passport, ID card, bill, check deposit, or credit card, which can then be used as a form of identification. The user begins by capturing an image of the information page. The captured image is sent for data extraction. The information is shown to the user, which they can verify, correct as needed and finally submit. Please note that when the Submit button is pressed in the Demo program, nothing happens.

Custom Component settings

The Custom Component has several settings that the user can change.

- **Camera settings** The user can choose from a variety of settings such as Gallery, Page Detection, Stability Delay and Roll Threshold.
- **Processing settings** The user can configure various settings related to the processing of images. Parameters like AutoCrop, Auto Rotate, Deskew, Scale (dpi), and Sharpen can be configured. Deskew by content or layout can also be configured.
- **Server settings** The user can configure the server to which the image will be submitted for extraction. The URL can be HTTP or HTTPS.
- **Edit component labels** The user can modify the text used for various field labels use on the screens.

Adding the license

In addition to the methods below, you can also use a bar code reader to capture the SDK license at run time.

For more information, see License capture control object.

Android

To add an Android license to the Kofax Mobile Demo Application with Android Studio:

1. After the project is imported into Android Studio, navigate to `Src/com.kofax.mobiledemo.common/License.java`.
2. Change the `PROCESS_PAGE_SDK_LICENSE` variable from "Invalid License" to your provided license.

iOS

To add an iOS license to the Kofax Mobile Demo Application:

1. Open the project in Xcode.
2. Navigate to `KofaxMobileDemo/KofaxSDKHelperFiles/EVRS/kfxEVRS_License.h`
3. Change the `PROCESS_PAGE_SDK_LICENSE` variable from "No License" to your provided license.

About the EasySnapApp application

The EasySnapApp sample application, included with the SDK, demonstrates the capabilities of the mobile SDK's Packaged Capture Experience. It has similar functionality to the Kofax Mobile Demo application, but much of the logic is encapsulated within the Packaged Capture workflows. Building an application this way requires significantly less effort than using the standalone SDK classes, but it does not give the developer as much control. Refer to the Packaged Capture Experience section of this document for more details.

For information on building the EasySnapApp, refer to the Getting Started with the SDK section of this document

Adding the EasySnapApp license

To add an Android license to the EasySnapApp with Android Studio:

1. After the project is imported into Android Studio, navigate to src\main\java\com\kofax\sdk\samples \easysnap\common\License.java.
2. Replace the SDK_LICENSE value with your provided license.

To add an Android license to the EasySnapApp for iOS:

1. Open the project in Xcode.
2. Navigate to EasySnapApp\AppDelegate.m.
3. . Replace @"MyLicenseString" with your provided license.