Kofax Mobile ID Verification

Release Notes

Version: 2.5.0.6

Date: 2020-09-15

KOFAX

Table of Contents

About this release

The software and documentation is available from the Kofax Fulfillment Site: https://delivery.kofax.com/. A representative from your company registers on this site to download the software and documentation.

If you are already a Kofax customer, contact your Kofax Professional Services Regional Manager to discuss and plan your upgrade.

If you are an existing customer, follow the instructions below to access the product for this release:

1. Log in to the Kofax Fulfillment Site (https://delivery.kofax.com/).
2. From the Your Software list, locate and select the product you want to download.
3. Follow the instuctions on the Fullfillment Site to complete your download.

The available packages include the software, documentation, and license keys for the release.

New customers will receive an email from Kofax after their product's purchase. The email will contain a serial number to use when registering on the Kofax Fulfillment Site. Registration provides customers with the credentials needed to download their product.

Product documentation

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

Default online documentation

The product documentation is available at the following location.

https://docshield.kofax.com/Portal/Products/en_US/KMC/3.5.0-cs5i340uk7/KMID.htm

Configure offline documentation

To access the documentation offline, download the following files:

- KofaxMobileIDDocumentation-2.5.0_EN.zip
- KofaxMobileIDVerificationDocumentation_2.5.0_EN.zip

Download these files from the Kofax Fulfillment Site and extract it on a local drive available to your users.

The compressed files include the `print` folder that contains all guides, such as the Installation Guide and the Administrator's Guide. The KofaxMobileIDDocumentation-2.5.0_EN.zip file also contains a `help` folder with API references.

New features

The following features were added for this release.

Customizable scripting for ID extraction

You can customize ID extraction logic through scripting. Field data can now by manipulated by using custom LUA scripts to extend or customize field validation logic. Script customizations run on both the server and on-device extraction. For more information on scripting, see Scripting for ID extraction.

New countries and document types

Mobile ID 2.5.0.6 expands support for more document types from the following countries:

Nation	Type	Years Supported
El Salvador	Driver License	unknown
	ID	2010, 2011
	Resident Card	unknown
Guatemala	Consular ID	2015
	Driver License	2009
	ID	2009
Japan	Driver License	2000
	Residence Card	2013
Morocco	ID	2008
Panama	Driver License	unknown
	Tribunal Electoral Card	2010
Russia	Passport	2006, 2010

The following document types were added to the *Kofax Mobile ID Extracted Field Tables Reference*.

Nation	Type	Years Supported
Australia - Western Australia	Driver License	2014
Canada	Permanent Resident Card	2016
Canada - Quebec	Driver License	2015
Denmark	Driver License	2015
European Union	Driver License (back for classification)	unknown
Hong Kong	Identity Card	2018
Hungary	Identity Card	2016
Ireland	Passport Card	2015

Nation	Type	Years Supported
Italy	Driver License	2013
Italy	Identity Card	2017
Lithuania	Driver License	2013
Mexico	Professional License	2015
Mexico - State of Mexico	Driver License	2017
New Zealand	Driver License	2014
Singapore	Work Permit	2017
Slovakia	Driver License	2013, 2015
Slovenia	Driver License	2013
Sweden	Driver License	2016
Turkey	Identity Card	2016

See the *Kofax Mobile ID Extracted Field Tables Reference* for the full list of supported document types.

New Spain Identification Card field

Spain identification cards have a new field, CardAccessNumber.

Resolved issues

This section describes issues resolved in the following releases.

Austria eResident Permit IssueDate not returned standard format

1201942: To extract the adjoining IssueDate and PlaceOfIssue fields from the Austria eResident Permit (2011), they were labeled as one line to be parsed into the two values. However, the IssueDate value was extracted as it appeared on the card in dd.mm.yyyy format and not in yyyy-mm-dd format, as needed.

ID Verification attempted to connect to external server

1086860: The Verification Application server attempted to validate the IP address of the server by connecting to an external geolocation service.

NLD ID 2014 variant returned PersonalNumber with check digit

1106574: When sending back or front and back Netherlands IDs, returning personal number appended by check digit next to the "personal number" in the MRZ.

Known issues

This section describes issues that you may encounter while using Kofax Mobile ID Capture. Work-around solutions are provided, as applicable.

ID Verification slower when Linux services are installed on 28-core systems

1287751: Kofax Mobile ID Verification is slower when installing Linux services on a 28-core system as compared to 8 cores.

Merchant Activation link does not work when installed with Docker

1280791: When installing the application server with Docker, the Merchant Activation link uses the internal port instead of the external port. The URL uses port 4434 instead of 443.

Workaround: Correct the URL to use the correct port and try the link again.

Names incorrectly concatenated for Michigan driver licenses

1099247: When a Michigan driver license is passed to the ID Capture project, using the front side alone produces results from OCR, where the first name and middle name are separated. But if the front and back sides of a Michigan ID are passed to the solution, the bar code on the back of the ID does not correctly separate the name fields and does not leave a space between items. Because bar code results take precedent, and are presumed correct, the project produces incorrect results.

Additional documentation

This section supplements the documentation provided with the product with changes, corrections, and additional information.

Scripting for ID extraction

You can now customize on-device extraction logic through scripting. Field data can now by manipulated by using scripts. This document shows how to update the Customization.script file in the `Models` folder.

These scripts use the LUA scripting language. This document does not provide instructions on using LUA. For information about LUA, see www.lua.org.

Specifying the script

The CombinedIDs file contains models for on-device extraction. This can either be downloaded from the server or pre-complied into the app. Use the method you choose to update the file and incorporate it into the app.

Get the CombinedIDs file first. The CombinedIDs model contain different regions (such as Canada, the United States, and Australia) with each region containing different variants. Each variant contains a Customization.script file. This script is used while performing validation.

SDK looks for these variant level script file paths in validation stage. The following figure shows the variant level script that users can apply to their own scripts.

When adding a customized script to your models, name it Customization.script.

Methods used in Customization.script

The Customization.script may contain the following methods for each field:

- preValidate<FieldName>
- validate<FieldName>
- postValidate<FieldName>

The field name is case-sensitive.

For example, for DateOfIssue, following methods can be used:

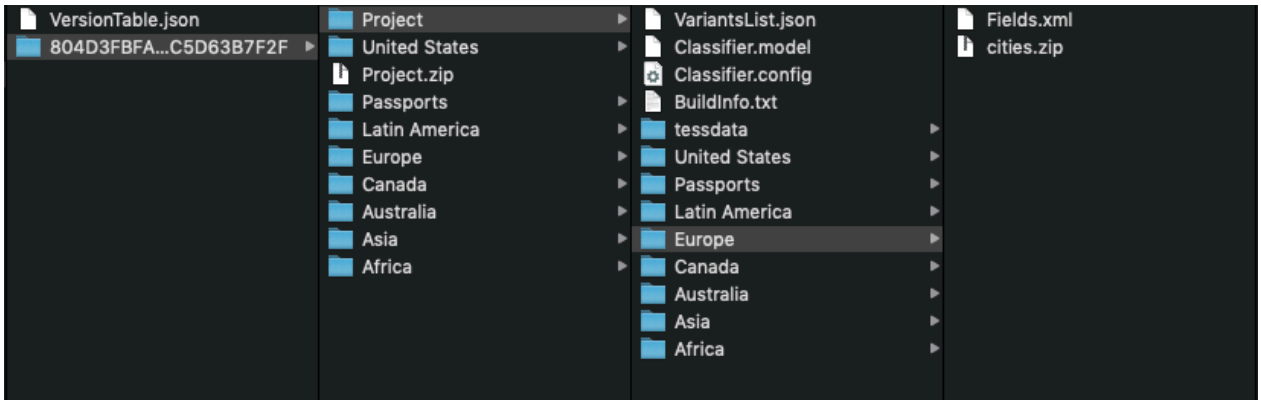
- preValidateDateOfIssue
- validateDateOfIssue
- postValidateDateOfIssue

Field names available in the variant

Users cannot define any new field name for output. Instead pre-existing fields must be used. Below are some of the available fields, which are common in every variant.

- FirstName
- MiddleName
- LastName
- FullName
- Gender
- ExpirationDate
- IssueDate
- DateOfBirth
- IDNumber
- License
- AddressLine1
- AddressLine2
- AddressLine3
- AddressLine4
- City
- State
- ZIP

Users can also check these fields from the Fields.xml file which is available in Project.zip file. The following figure shows the Field.xml file.



The SDK looks for preValidation, validation and postValidation methods in the script for each field.

These methods will be helpful for the parsing the fields like address, date, and name.

Order of execution of the methods in a script

The SDK does the following when an image is received:

1. Processing: The SDK processes the image in this stage.
2. Classification: This stage classifies the variant type (such as WI_2015_u21, WI_2015_id and TN_2012.).
3. Extraction: This stage extracts the fields available on the ID card.
4. Validation: At this stage, the user has access to the fields available on the ID. The SDK looks for the Customization.script file, which is available at the level of the classified variants.

In the validation stage, events are executed as follows:

1. All pre-Validation field methods will be triggered before Validation stage.
2. Validation field methods will be executed.
3. Post-validation field methods will be executed.

For example, in the addressLine1 field, the validateAddressLine1 method may have parsing logic, parsing addressline into city, state and ZIP fields. The postValidateAddressLine1 method may have logic of boosting confidence logic.

Script example

The following example shows how to write a script to prepare IssueDate format. The format present on the document is DD.MM.YYYY, and we want to convert this into YYYY-MM-DD. The following validateIssueDate function uses a parameter called fields. This is a data type table used to parse the IssueDate. The script converts the fields into the desired format.

```
function validateIssueDate (fields)
 local issueDateIndex = -1
 for key, value in pairs(fields) do
  if value.label == "IssueDate" then
   issueDateIndex = key
   break
  end
 end

 if issueDateIndex >= 0  then
  local issueDateValue = fields[issueDateIndex].value
  print(issueDateValue)
  date, month, year = string.match(issueDateValue, '(%d%d).(%d%d).(%d%d%d%d)')  --

  if year ~= nil and month ~= nil and date ~= nil then

   formattedDate = string.format("%s-%s-%s", year,month,date)
   return {[0] = {index = issueDateIndex, label = "IssueDate", value = formattedDate}}
  end
 end
end
```

Full Customization.script example

The following is a sample Customization.script in its entirety.

```
function validateFirstName(fields)
 --Get FirstName field
 local firstNameFieldIndex = -1

 for key, value in pairs(fields) do
  if value.label == "FirstName" then
   firstNameFieldIndex = key
   break
  end
 end

 if firstNameFieldIndex >= 0 then
  local firstNameValue = fields[firstNameFieldIndex].value
  local replaceSmall = string.gsub(firstNameValue, "a", "*")
  local replaceCapital = string.gsub(replaceSmall, "A", "*")
  local finalFirstName = {index = firstNameFieldIndex, label = "FirstName", value =
 replaceCapital}
  return {[0] = finalFirstName}
 end
end

function validateLastName(fields)
 local LastNameFieldIndex = -1

 for key, value in pairs(fields) do
  if value.label == "LastName" then
   LastNameFieldIndex = key
   break
  end
 end

 if LastNameFieldIndex >= 0 then
  local lastNameValue = fields[LastNameFieldIndex].value
  local replaceSmall = string.gsub(lastNameValue, "a", "*")
  local replaceCapital = string.gsub(replaceSmall, "A", "*")
  local finalLastName = {index = LastNameFieldIndex, label = "LastName", value =
 replaceCapital}
  return {[0] = finalLastName}
 end
end

function validateState(fields)
 local stateFieldIndex = -1

 for key, value in pairs(fields) do
  if value.label == "State" then
   stateFieldIndex = key
   break
  end
 end

 if stateFieldIndex >= 0 then
  local stateValue = fields[stateFieldIndex].value
  stateTable = {}
  stateTable["AL"] = "Alabama"
  stateTable["AK"] = "Alaska"
  stateTable["AZ"] = "Arizona"
  stateTable["AR"] = "Arkansas"
```

```
  stateTable["CA"] = "California"
  stateTable["CO"] = "Colorado"
  stateTable["CT"] = "Connecticut"
  stateTable["DE"] = "Delaware"
  stateTable["FL"] = "Florida"
  stateTable["GA"] = "Georgia"
  stateTable["HI"] = "Hawaii"
  stateTable["ID"] = "Idaho"
  stateTable["IL"] = "Illinois"
  stateTable["IN"] = "Indiana"
  stateTable["IA"] = "Iowa"

  if tonumber(string.len(stateValue)) >= 2 then

   stateCode = string.sub(stateValue, 1, 2)
   print(stateCode)
   fullStateName = stateTable[stateCode]

   if fullStateName == nil  then
    print("state code not found")
   else
    local stateFieldTable = {index = stateFieldIndex, label = "State", value =
 fullStateName}
    return {[0] = stateFieldTable}
   end
  end
 end
end


function postValidateSponsor(fields)  --fields will be passed in the form of table

 --Get FirstName field
 local firstNameFieldIndex = -1

 for key, value in pairs(fields) do
  if value.label == "FirstName" then
   firstNameFieldIndex = key
   break
  end
 end

 --Get LastName field

 local LastNameFieldIndex = -1
 for key, value in pairs(fields) do
  if value.label == "LastName" then
   LastNameFieldIndex = key
   break
  end
 end


 if firstNameFieldIndex >= 0 and LastNameFieldIndex >= 0 then

  local firstNameValue = fields[firstNameFieldIndex].value
  local LastNameValue = fields[LastNameFieldIndex].value
  print(firstNameValue)
        print(LastNameValue)
  sponserName = firstNameValue .. " " .. LastNameValue
        print(sponserName)
  local sponserName = {index = firstNameFieldIndex, label = "Sponsor", value =
 sponserName}
  return {[0] = sponserName}
```

```
 end
end

function postValidateSponsorBranch(fields)  --fields will be passed from the SDK in the
 form of table

 --Get LastName field

 local LastNameFieldIndex = -1
 for key, value in pairs(fields) do
  if value.label == "LastName" then
   LastNameFieldIndex = key
   break
  end
 end

 if LastNameFieldIndex >= 0 then

  local LastNameValue = fields[LastNameFieldIndex].value
  SponsorBranch = string.len(LastNameValue)
        print(SponsorBranch)
  local SponsorBranch = {index = LastNameFieldIndex, label = "SponsorBranch", value =
 SponsorBranch}
  return {[0] = SponsorBranch}
 end
end
```