Business Fraud_Proof

BARCELONA PARIS SÃO PAU

iOS-SDK
IdCloud

Reference Manual

Version 1.5.18
5th of April, 2017

ICAR_
Ronda Can Fatjó, 21_ Parc Tecnològic del Vallès_ 08290 Cerdanyola del Vallès_ Barcelona.
Tel_ +34935942474 Fax_ +34935942479 www.icarvision.com_icar@icarvision.com

CONTENTS

1. INTRODUCTION 4
1.1 REQUIREMENTS 4
1.2 INTEGRATING THE SDK INTO YOUR PROJECT 5
1.3 COPYRIGHT MESSAGE 5

2. IMAGE CAPTURE MODULE 6
2.1. DATA STRUCTURE WITH CAPTURE CONFIGURATION 6
2.2. DATA STRUCTURE WITH ACQUIRED IMAGES 8
2.3. ICARCAPTURE_VC FOR IMAGE ACQUISITION 10

3. IDCLOUD CONNECTION MODULE 15
3.1. CONFIGURATION PARAMETERS 15
3.2. IDCLOUD SERVER CONNECTION FOR IMAGE PROCESSING 16
3.3. IDCLOUD PROCESS RESULT 19
3.4. CONFIRM SERVER CONNECTION CREDENTIALS 23

4. OFFLINE PROCESSING MODULE 25
4.1. ICARCAPTURE_VC FOR OFFLINE PROCESSING 25
4.2. ICAR_PAYMENT SERVER 29
4.3. CONFIGURATION FOR THE ICAR_PAYMENT SERVER CONNECTION. 29
4.4. CONNECTION TO ICAR_PAYMENT SERVER TO MAKE THE PURCHASE 30
4.5. PURCHASE PROCESS RESULT 34

5. QUICK REFERENCE GUIDE 36
5.1. EXAMPLE_CAPTURE FRONT/BACK/FACE AND SEND TO ICAR CLOUD SERVER 36
5.2. EXAMPLE_OFFLINE_PROCESSES 40
5.3. CUSTOMIZE AUTOFOTO PROCESS 41
5.4. SENDING CAPTURED IMAGES WITHOUT USING IDLCOUD CONNECTION 42

6. APPENDIX 43
6.1. TABLE WITH LITERALS TO BE TRANSLATED 43

7. CHANGES HISTORY **44**

1. INTRODUCTION

This document is intended for app developers on the iOS platform (iPhone and iPad devices).

The SDK-IDCloud, explained in this document, allows app developers to quickly and easily integrate IDCloud server connections in order to analyze documents. Its interface abstracts the management of web services, such as the parsing of results obtained in XML format.

In addition to a remote server connection, it offers developers an interface to manage the acquisition of images in the user terminal without having to access the hardware coding involved.

Finally, SDK-IDCloud provides a series of off-line processes for documents. Given that in order to allow these processes, one must first connect to the Icar server. The interface also abstracts the management of the web services involved in this task.

This SDK provides the developer with a static library with a group of classes and resources (all grouped in a framework file), the functions of which may be divided into three modules:

1. Image Capture Module.
2. IDCloud Connection Module.
3. Offline Process Module.

If you use module 1 to send the captured data to your server, without using module 2, please refer to section 5.4 for more information on how to perform this process.

The size of sdk is detailed below:

	IcarSDK.framework	How much does the ipa increase using the sdk?	How much does the apk increase in the user device using the sdk?
Size	15 MB	8 MB	15 MB

Finally, in demo mode, two demo projects are provided that explains the interface of the three modules (see section 5.1. EXAMPLE_CAPTURE FRONT/BACK/FACE AND SEND TO ICAR CLOUD SERVER and 5.2. EXAMPLE_OFFLINE_PROCESSES for more information about it).

1.1 REQUIREMENTS

- XCode 6 and iOS SDK 8.

- Devices with ARMv7, ARMv7s, and ARM64 architecture. It may also work with the simulator, but obviously cannot acquire images through the camera.
- iPhone and iPad of any size and resolution.

1.2 INTEGRATING THE SDK INTO YOUR PROJECT

In order to integrate the library into your XCode project, complete the following:

1. Add framework "SDKRoot/SDK/iphoneos/IcarSDK.framework to your project.
2. Check that this framewrok appears on General/Embedded Binaries & Linked Frameworks and libraries.
3. Build Settings, disable bitcode (Enable Bitcode = No)
4. info.plist add new entry "Privacy - Camera Usage Description". For example = " This app use the camera in order to allow the user capture and digitalize the card. Our solutions is focused in reads, extracts the data and verify the user cards."
5. In the provided library there is a group of non-translated literals; it is best to translate them through the *Localizable.strings* of the application being developed.
    - You will find the identifier/definition of each literal in section “6.1. TABLE WITH LITERALS TO BE TRANSLATED” of this document.

1.3 COPYRIGHT MESSAGE

2. IMAGE CAPTURE MODULE

As a complement to the IDCloud server connection, iOS-SDK offers a customized IcarCapture_VC class for accurate acquisition of images on mobile phones.
This is in addition to acquiring images to be processed by the server, as shown in module “3. IDCLOUD CONNECTION MODULE”; this class also offers the interface for carrying out offline processes (see “4. OFFLINE PROCESSING MODULE”).
In order to view the *View Controller* associated with the IcarCapture_VC class, use a customized UINavigationController (IcarCapture_NC.h) and the following auxiliary classes:

- IcarCapture_Configuration.h:
  In this version, 1.5.18, the data structure contains the configurable field *maxImageSize*, which will determine the maximum megapixel size that a captured image will have. By default its value is 2, and if no size modification is desired, this must be changed to -1.
  In addition, the type of process to be carried out must be specified (document capture; obtain MRZ or PDF417 information, etc.). This data structure must be introduced as a parameter during start-up of the IcarCapture_VC *View Controller.*
  Important: In version 1.5.18 all acquisition requires a valid Icar license. The license information can be specified using the "icarLicenseKey" field (for more information contact the sales team of Icar).

- IdCloud_LocalImages.h:
  This data structure must be introduced as a parameter during start-up of the IcarCapture_VC *View Controller*, and once the *VC* is finished you can get the captured image through its interface.

Important: If you decide to capture documents (maybe also the selfie) but without using IDCloud_Connection, please see section "5.4. SENDING CAPTURED IMAGES WITHOUT USING IDLCOUD CONNECTION" above for more information on what information to send.

These two classes will be detailed below, and the IcarCapture_VC class interface will be explained at the end of this section.

2.1. DATA STRUCTURE WITH CAPTURE CONFIGURATION

The device’s image capture process can be configured using an instance of the IcarCapture_Configuration class.

Physical location:
IcarCapture_Configuration.h

Declaration:
```
@interface IcarCapture_Configuration : NSObject
{}
```

```
…(…, readwrite) int maxImageSize;   //default: 2
…(…, readwrite) bool captureDocWithAutoFoto;   //default: true
…(…, readwrite) bool liveness;   //default: true
…(…, readwrite) bool enableVibrationMRZ;   //default: true
…(…, readwrite) bool enableVibrationAutofo;   //default: true

…(…, readwrite) bool enableVibrationFaceDetection;   //default: true
…(…, readwrite) bool autoFotoWithManualButton;   //default: false


- (void) setInternalProcess:(ICAR_CaptureProcess) internalProcess;
- (ICAR_CaptureProcess) getInternalProcess;
```

Properties:

maxImageSize	Maximum image size in megapixels (documents to be processed by the IDCloud server): - By default its value is 2 megapixels. - If no modification to the acquired image is desired, you must assign this number to -1.
enableVibrationMRZ enableVibrationAutoFoto enableVibrationFaceDetection	enables or disbales mobile vibration on processes MRZ, AutoFotot and FaceDetection.
autofotoWithManualButton	enable button to capture doc. Appear only if captureDocWithAutoFoto is true.
icarLicenseKey	Important: All acquisition functionality requires a valid Icar license.

setInternalProcess Function:

This function allows you to specify the type of process to be carried out locally on the phone when custom view images are acquired. In order to acquire images to be processed by the IdCloud server, you must use CaptureProcess_CAPTURE_IMAGE.

Parameter type	Name	Parameter definition
ICAR_CaptureProcess*	internalProcess	Type of process to be captured: • CaptureProcess_CAPTURE_IMAGE: front or back of the image to be captured. To send to the IDCloud server. It is not necessary to have local credits available (credits are explained in section "4. OFFLINE PROCESSING MODULE"). This process can be with or without autofoto (using captureDocWithAutoFoto property). • CaptureProcess_CAPTURE_MRZ: capture the MRZ string of a document. It is necessary to have credits available (explained in section "4.4. CONNECTION TO ICAR_PAYMENT SERVER TO MAKE

		THE PURCHASE"). • CaptureProcess_CAPTURE_PDF417: capture the PDF417 string of a document. It is necessary to have credits available. • Capture_FACE_DETECTION: automatically capture the face of the user. It is not necessary to have local credits available. This process can be with or without liveness (using liveness property). • CaptureProcess_NONE: default value.

(*Physical location of enum ICAR_CaptureProcess: IcarSDK_Defs.h)

getInternalProcess Function:

This function allows you to specify the type of process to be carried out locally on the phone when custom view images are acquired.

2.2. DATA STRUCTURE WITH ACQUIRED IMAGES

As explained in section "2.3. ICARCAPTURE_VC FOR IMAGE ACQUISITION", you can use IcarCapture_VC to acquire an image from the device without having to obtain camera access. To do this, an instance of ICloud_LocalImages must be set as a start-up parameter, and when closing the *View Controller* you can obtain the result of the capture.
This data structure also allows images to be loaded without using ICarCapture_VC, for example acquiring them by using *UIImagePickerController*.

Physical location:

IdCloud_ LocalImages.h

Declaration:

```
@interface IdCloud_LocalImages : NSObject{}

//---Functions:
- (NSString*)    getImage_base64:(int) _imgPos;
- (UIImage*)     getImage:(int) _imgPos;
- (BOOL)         setImage: (UIImage*)_img
                 withConf:(IcarCapture_Configuration*) _config
                 andPos:(int) _imgPos;

- (int)          getNumLoadedImages;
- (BOOL)         removeImage:(int) _pos;
- (NSString*)    getDeviceInfo:(int) _imgPos
                      withConf:( IdCloud_Configuration*)_conf;

…(…, readwrite) IcarParsedMrz* parsedMRZ;
…(…, readwrite) IcarParsedPdf417* parsedPDF417;
…(…, readwrite) UIImage *faceDetected;
```

Properties:

parsedMRZ	Data structure with the following information: • mrzLines (NSString): MRZ lines of the document. • globalIntegrity (NSString): global integrity test. • fieldsIntegrity (NSString): integrity tests for various MRZ fields. • fields (NSString): All (locally) parsed fields from the mrzLines.
parsedPDF417	Data structure with information contained in PDF417 bar codes. This information is provided in NSString and NSData formats.
faceDetected	Selfie acquired automatically in UIImage format.

setImage Function:

With this function, a conversion of the past image (front or back of the document) to base64 will be done as a parameter. Additionally, keeping in mind the maxImageSize property of the *_config* entry parameter, the image size (in megapixels) will be adjusted.

Parameter type	Name	Parameter definition
UIImage*	_img	Image to be processed
IcarCapture_Configuration*	_config	Configuration used to process images, in particular the maxImageSize property. If no modification of the image is desired, set value to -1. By default its value is 2.
int	_imgPos	Position (0 or 1) in which the acquired image will be stored.

getImage_base64 Function:

Returns the image in base64 (*NSString**) to the _imgPos position. If there is no stored image at this position, @"" will be returned.

Parameter type	Name	Parameter definition
int	_imgPos	Position (0, 1) of the image to be obtained in base64.

getImage Function:

Function that provides the (UIImage*) image from the _imgPos position. If there is no stored image at this location, the result will be 0.

Parameter type	Name	Parameter definition
int	_imgPos	Position (0, 1) of the image to be obtained in base64.

getNumLoadedImages Function:

Function that returns that number of images (front or back of the document) that have been processed: 1, 2, or none.

removeImage Function:

This function allows us to remove the image that was captured locally and stored in position 0 or 1.

Parameter type	Name	Parameter definition
int	_pos	Position (0, 1) of the image to be eliminated in base64.

getDeviceInfo Function:

This function return part of the xml string that the webservice send to the server IDCloud to process the document (using IDCloudConnection). In particulary this part of xml is that called deviceInfo. Please refer to the documentation call " /SDKRoot/Documentation/IDFast_IDFraud_Result_Fields_EN.pdf" for more information about this xml.

Parameter type	Name	Parameter definition
int	_imgPos	Position (0, 1) of the image to be obtained in base64.
IdCloudConfiguration	_config	Configuration of idcloud connection.

Example:

```
IdCloud_LocalImages*    idCloud_LocalImages;
UIImage* img1 = [UIImage imageNamed:@"foto1.JPG"];
UIImage* img2 = [UIImage imageNamed:@"foto2.JPG"];
IcarCapture_Configuration *config = [[IcarCapture_Configuration alloc] init];
idCloud_LocalImages = [[IdCloud_LocalImages alloc] init];
[idCloud_LocalImages setImage:img1
                     withConf: config andPos:0];
[idCloud_LocalImages setImage:img2
                     withConf: config andPos:1];
```

*Note: In this sample code, the images of project resources are acquired. Now, the application can acquire them using a UIImagePickerController, an AVCaptureSession, or additionally through use of the IcarCapture_VC detailed below.

2.3. ICARCAPTURE_VC FOR IMAGE ACQUISITION

Physical location:

IcarCapture_VC.h

Declaration:

```
@protocol IcarCapture_Delegate <NSObject>
@required
- (void) captureDidCancel:(ICAR_CaptureResult) _result;
- (void) captureDidComplete;
@end


@interface IcarCapture_VC : UIViewController
{}

- (id)initWithDelegate:(id<IcarCapture_Delegate>)_delegate
      andConfiguration:(IcarCapture_Configuration*) _conf
         andLocalImage:(IdCloud_LocalImages*)_images
           andPosImage:(int) _imgPos;
```

Properties:

No Properties.

initWithDelegate Function:

The instance of the IcarCapture_VC class must be initialized through this function. An instance of the IdCapture_Configuration class and an instance of the IdCloud_LocalImages class are used to store the result, the position (front or back) of the image to be acquired, and a delegate to determine when the capture process has finished:

- captureDidCancel Function: This will communicate the cancellation, by the user, of the image capture process. If there are insufficient credits to carry out certain processes, you will also receive a notification indicating that this is the case. The possible values of ICAR_CaptureResult (defined in IcarSDK_Defs.h) are:
  - CaptureResult_CANCELED_BY_USER.
  - CaptureResult_NO_CREDIT.
  - CaptureResult_LICENSE_PDF417_ERROR.
  - CaptureResult_INVALID_ICAR_LICENSE_KEY.
  - CaptureResult_ERROR.
- captureDidComplete Function: This will communicate the completion of the image capture process by the user. Using parameter _images, the image acquired will be stored at position _imgPos or the information (NSString*) processed offline, e.g. an MRZ.

Parameter type	Name	Parameter definition
IcarCapture_Configuration*	_conf	Structure for process configuration. In this version, only one of its properties will be used: the maxImageSize field. This field defines the maximum number of megapixels used to acquire the image on the device. This is described in detail in section "2.1. DATA STRUCTURE WITH CAPTURE CONFIGURATION" of this document.
Id<IcarCapture_Delegate>	_delegate	Delegate that will be used for notification of capture process completion.
IdCloud_LocalImages	_images	Data structure where the images acquired by the terminal (or information in NSString* format) will be stored. This is described in detail in section "2.2. DATA STRUCTURE WITH ACQUIRED IMAGES" of this document.
int	_imgPos	Index (0 or 1) in which the acquired image will be stored. The 0 index may be used to refer to the front or back interchangeably.

Example 1 (Capture Doc / CaptureProcess_CAPTURE_IMAGE):

In file .h you must convert to the *ViewController* as delegate of the IcarCapture_Delegate. It is important to keep the life cycle of the IcarCapture_Configuration and IdCloud_LocalImages instances at the same level as their containing class.

```
@interface IdCloud_VC : UIViewController <IcarCapture_Delegate>
```

```
{
    IdCloud_LocalImages*        m_IdCloud_LocalImages;
    IcarCapture_Configuration*  m_IcarCapture_Config;
}
```

Start up the structures.

```
- (void)viewDidLoad
{
   [super viewDidLoad];
   m_IdCloud_LocalImages = [[IdCloud_LocalImages alloc] init];
   m_IcarCapture_Config  = [[IcarCapture_Configuration alloc] init];
   [m_IcarCapture_Config setInternalProcess:CaptureProcess_CAPTURE_IMAGE];
   m_IcarCapture_Config.icarLicenseKey = @"2ZZPUjJ72E8+...";
}
```

The following code shows the execution of the image capture module.

```
- (IBAction)showImageCapture:(id)sender
{
    IcarCapture_VC*    vc    =    [[IcarCapture_VC    alloc]    initWithDelegate:self
andConfiguration:m_IcarCapture_Config           andLocalImage:m_IdCloud_LocalImages
andPosImage:0];

    IcarCapture_NC *nav = [[IcarCapture_NC alloc] init];
    [nav setViewControllers:[NSArray arrayWithObjects:vc, nil]];
    [self presentViewController:nav animated:NO completion:nil];
}
```

Through the *captureDidCancel* and *captureDidComplete* functions, you can receive notifications regarding the final capture result. It is important to execute *dismissViewControllerAnimated* in these functions to make the IcarCapture_VC disappear.

```
#pragma mark - IcarCapture_Delegate
- (void) captureDidCancel
{
    if (_result == CaptureResult_CANCELED_BY_USER)
    {
        [self dismissViewControllerAnimated:NO completion:nil];
    }
    //else if (....)
    else
    {
        NSLog(@"MainDemo_TVC:ICAR_CaptureResult=CaptureResult_NO_CREDIT");
    }
}
- (void) captureDidComplete
{
    //In this example, we will insert the image into a UIViewImage
    _img.image = [m_IdCloud_LocalImages getImage:0];
    [self dismissViewControllerAnimated:NO completion:nil];
}
```

Illustration 1: IcarCapture_VC screen capture (mode `CaptureProcess_CAPTURE_IMAGE`, with Autofoto and autofotoWithManualButton).

Illustration 2: IcarCapture_VC screen capture (mode `CaptureProcess_CAPTURE_IMAGE`, without Autofoto).

Example 2 (Capture Face / CaptureProcess_CAPTURE_FACE_DETECTION):

To capture the face, we have to proceed as in Example 1. We just have to change the call to setInternalProcess function using as argument the value CaptureProcess_CAPTURE_FACE_DETECTION.
In addition you can use IcarCapture_Configuration.liveness property to specify the capture in normal mode or using a liveness test. By defualt its value is true.

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    m_IdCloud_LocalImages = [[IdCloud_LocalImages alloc] init];
    m_IcarCapture_Config  = [[IcarCapture_Configuration alloc] init];
    [m_IcarCapture_Config setInternalProcess:CaptureProcess_CAPTURE_FACE_DETECTION];
    m_IcarCapture_Config.liveness = true;
    m_IcarCapture_Config.icarLicenseKey = @"2ZZPUjJ72E8+...";
}
```

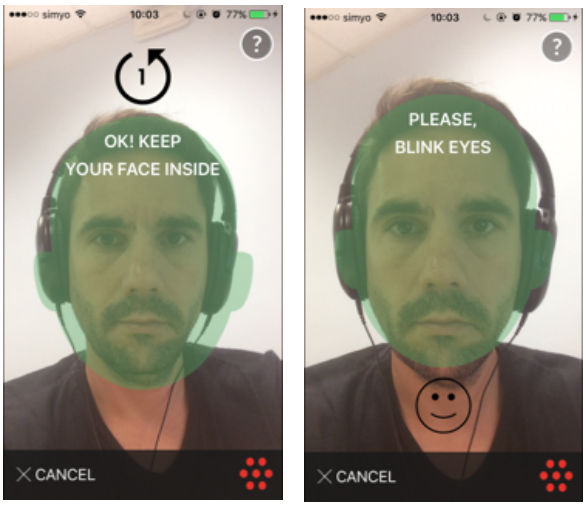

Illustration 3: Left image using normal mode, right image using liveness.

3. IDCLOUD CONNECTION MODULE

This module enables connection to an IDCloud server in order to read documents, obtaining, if successful, a data structure with the processed information.

3.1. CONFIGURATION PARAMETERS

First, enter the necessary information to connect to the IDCloud server. The IdCloud_Configuration class is available for this.

Physical location:

IdCloud_Configuration.h

Declaration:

```
@interface IdCloud_Configuration : NSObject{}
…(…, readwrite) NSString* server_Port;
…(…, readwrite) NSString* server_URL;
…(…, readwrite) int       server_Timeout;
…(…, readwrite) NSString* user;
…(…, readwrite) NSString* password;
…(…, readwrite) NSString* company;
…(…, readwrite) NSString* icarFaceValidation;
…(…, readwrite) NSString* emailToCheck;
…(…, readwrite) NSString* phoneNumberToCheck;
…(…, readwrite) NSString* icarLicenseKey;
…(…, readonly) NSString*  version;


- (void) setFaceValidation:(UIImage*)image;
```

Properties:

server_Port	Port number for the connection to the IDCloud server.
server_URL	URL where the IDCloud server is found.
server_Timeout	Maximum wait time without receiving a response from the IDCloud server. Expressed in milliseconds. By default its value is 2 seconds.
user	Credential with user information used to identify oneself to the IDCloud server.
password	Credential with password information used to identify oneself to the IDCloud server.
company	Credential with the company information used to identify oneself to the IDCloud server.
version	Version of the libIcarSDK_IdCloud.a library (readonly).
icarFaceValidation	Selfie of the document user. Once the front of the document and the selfie are sent, IDCloud will return a comparison test between the two faces.
emailToCheck	Email acount to be validated by idcloud server.
phoneNumberToCheck	Phone number to be validated by idcloud server. It's work only

	with spanish numbers and the format of this must be 34xxxxxxxx
icarLicenseKey	Important: All connections to IDCloud Server require a valid Icar license (for more information contact the sales team of Icar).

setFaceValidation Function:
This function allows us to send the selfie to be compared with extracted image of document.

Parameter type	Name	Parameter definition
UIImage*	image	Image to be sent to iCloud server. This image will be used as selfie to be compared with the extracted image of document.

Example:

```
IdCloud_Configuration*  idCloud_Conf;
idCloud_Conf = [[IdCloud_Configuration alloc] init];
idCloud_Conf.user            = @"user";
idCloud_Conf.password        = @"1234567";
idCloud_Conf.company         = @"icar";
idCloud_Conf.server_Port     = @"50061";
idCloud_Conf.server_Timeout  = 20000; //in milliseconds
idCloud_Conf.server_URL      = @"https://demoidcloud.icarvision.com";
idCloud_Conf.icarLicenseKey  = @" HALXJtAHO...";
```

3.2. IDCLOUD SERVER CONNECTION FOR IMAGE PROCESSING

Once the connection data has been configured and the IdCloud_LocalImage data structure has been completed, the IDCloud server request may be sent in order to process the documents to be sent. The IdCloud_Configuration class is available for this.

Physical location:
IdCloud_Connection.h

Declaration:

```
@protocol IdCloud_Connection_Delegate <NSObject>
@required
- (void) connectionDidCancel:(IdCloud_Result*)_result;
- (void) connectionDidComplete:(IdCloud_Result*)_result;
@end


@interface IdCloud_Connection : NSObject{}

//---Functions:
- (void) processIdCloudWithConf:(IdCloud_Configuration*)_config
                         images:(IdCloud_LocalImages*) _images
                       delegate:(id<IdCloud_Connection_Delegate>)_delegate
                  andDataResult:(IdCloud_DataResult*)_dataResult;
```

```
- (void) checkIdCloudWithConf:(IdCloud_Configuration*)_config
                     delegate:(id<IdCloud_Connection_Delegate>)_delegate;
```

*Note: The checkIdCloudWithConf function is detailed in section “ 3.4. CONFIRM SERVER CONNECTION” of this document.

processIdCloudWithConf Function:

The process request is sent to the server through this function. The IdCloud_Connection_Delegate protocol will be used to receive the response through a delegate:

- connectionDidCancel function: The request failed. The error description will be received through the IdCloud_Result entry parameter (detailed below).
- connectionDidComplete function: The request was sent successfully. The data can be found in the IdCloud_DataResult data structure. This will be passed into the processIdCloudWithConf function as a parameter.

Parameter type	Name	Parameter definition
IdCloud_Configuration*	_config	Configuration with which to make the connection.
IdCloud_LocalImages*	_images	Structure with the images already processed by the phone.
id<IdCloud_Connection_Delegate>	_delegate	Delegate with which to communicate if the connection is successful (connectionDidComplete) or not (connectionDidCancel).
IdCloud_DataResult*	_dataResult	Structure with the processed information. Explained in "3.3. IDCLOUD PROCESS RESULT".

*Note: In the following section, "3.3. IDCLOUD PROCESS RESULT“, the IdCloud_DataResult structure will be explained, which will contain the information processed by the IdCloud server. In addition, this section will explain the IdCloud_Result data structure.

Example:

In file .h you must convert to the *ViewController* as delegate of the IdCloud_Connection_Delegate. It is important to keep the life cycle of the IdCloud_Connection instance at the same level as its containing class.

```
@interface ViewController: UIViewController <IdCloud_Connection_Delegate>
{
    IdCloud_DataResult*  m_DataResult;
    IdCloud_Connection*  m_IdCloud_Connection;
}
```

The following function shows the complete code to make the connection.

```
- (void) doConnection
{
//Configure the server information:
IdCloud_Configuration*  idCloud_Conf;
```

```
idCloud_Conf = [[IdCloud_Configuration alloc] init];

idCloud_Conf.user           = @"user";
idCloud_Conf.password       = @"1234567";
idCloud_Conf.company        = @"icar";
idCloud_Conf.server_Port    = @"50061";
idCloud_Conf.server_Timeout = 20000; //in milliseconds
idCloud_Conf.server_URL     = @"https://demoidcloud.icarvision.com";
idCloud_Conf.icarLicenseKey = @"KHJHE3jekeh3iu...";

//Configure the images to be sent to the server:
IdCloud_LocalImages*    idCloud_LocalImages;
UIImage* img1 = [UIImage imageNamed:@"doc1.JPG"];
UIImage* img2 = [UIImage imageNamed:@"doc2.JPG"];
UIImage* img3 = [UIImage imageNamed:@"selfie.JPG"];

IcarCapture_Configuration *config = [[IcarCapture_Configuration alloc] init];

idCloud_LocalImages = [[IdCloud_LocalImages alloc] init];
[idCloud_LocalImages setImage:img1
                      withConf:config andPos:0];
[idCloud_LocalImages setImage:img2
                      withConf:config andPos:1];

[idCloud_Conf setFaceValidation:img3];
//Execute the process in the server:
m_DataResult = [[IdCloud_DataResult alloc] init];

m_IdCloud_Connection = [[IdCloud_Connection alloc] init];
[m_IdCloud_Connection processIdCloudWithConf:idCloud_Conf
                                      images:idCloud_LocalImages
                                    delegate:self
                                andDataResult:m_DataResult];
}
```

The connectionDidComplete function must be implemented in order to be notified that the connection was successful. The information sent by the server will be obtained through an instance of the IdCloud_DataResult class.

```
- (void) connectionDidComplete:(IdCloud_Result*)_result
{
    if (m_DataResult != nil)
    {
        NSLog(@"%@",m_DataResult.xml);
    }
}
```

Finally, the connectionDidCancel function must be implemented in order to be notified that the connection was **not** completed successfully. The IdCloud_Result data structure allows you to find out what kind of error occurred using the *error* (enumerate) property, and if it is a ServerConnection error, there will be a description in the *error_desc* field and an *error_code*.

```
#pragma mark - IdCloud_Connection_Delegate
- (void) connectionDidCancel:(IdCloud_Result*)_result
{
    switch (_result.error)
    {
        case IdCloudError_ConnectionNull:
            NSLog(@"Error: Connection Null");
            break;
        case IdCloudError_InvalidConfig:
```

```
            NSLog(@"Error: IdCloudError_InvalidConfig");
            break;
        case IdCloudError_InvalidImages:
            NSLog(@"Error: IdCloudError_InvalidImages");
            break;
        case IdCloudError_ServerConnection:
            NSLog(@"Error: IdCloudError_ServerConnection");
            NSLog(@"connectionDidCancel_desc: %@",_result.desc);
            break;
        default:
            break;
    }
}
```

3.3. IDCLOUD PROCESS RESULT

While executing the processIdCloudWithConf function as shown in section "3.2. IDCLOUD SERVER CONNECTION FOR IMAGE PROCESSING", you must transfer an instance of the IdCloud_DataResult class. The instance of this class will be filled in with the information processed by the IDCloud server.

The information obtained through this process will be in XML format. For more information about the XML format and its possible values, consult "SDKRoot/Documentation/IDFast_IDFraud_Result_Fields_EN.pdf" (Entity "DocumentCheckOutV2").

In addition to offering the XML in NSString* and NSData* format (in order to be able to be manipulated by any parsing API), two functions are available to obtain the values for any grouping of search fields. To do this, it uses the IdCloud_DataResult_Delegate protocol and some delegates which will receive the information asynchronously once the parsing is completed (the fieldsContentDidComplete and imagesContentDidComplete functions).

Physical location:

IdCloud_DataResult.h

Declaration:

```
@protocol IdCloud_DataResult_Delegate <NSObject>
@optional
- (void) fieldsContentDidComplete:(NSDictionary*)_results;
- (void) imagesContentDidComplete:(NSDictionary*)_results;
- (void) imagesContentDidCancel:(NSDictionary*)_errorDesc;
- (void) imagesContentDidCancel:(NSDictionary*)_errorDesc;
@end


@interface IdCloud_DataResult : NSObject{}

//---Properties:
…(…, readonly) NSString*    xml;
…(…, readonly) NSData*      xmlData;

//---Functions:
- (void) getAllFields:(id<IdCloud_DataResult_Delegate>) _delegate;

- (void) getFieldsContent:(NSArray*)_codes
             andDelegate:(id<IdCloud_DataResult_Delegate>) _delegate;
```

```
- (void) getResultImages:(NSArray*)_codes
             andDelegate:(id<IdCloud_DataResult_Delegate>) _delegate;
```

Properties:

xml	The XML processed by the server in NSString format.
XmlData	The XML processed by the server in NSData format.

getAllFields Function:

Through this function, all values (*NSString**) are obtained in the fields (*NSString**) contained in the XML. The IdCloud_DataResult_Delegate protocol will be used to receive the response (once parsed):

- fieldsContentDidComplete Function: This will communicate that the parsing has finished and an *NSDictionary* will be obtained with values and fields, such as *key*.

Parameter type	Name	Parameter definition
Id<IdCloud_DataResult_Delegate>	_delegate	Delegate that will be used for notification of parsing completion through the fieldsContentDidComplete function.

getFieldsContent Function:

Through this function, you can search to find the values (*NSString**) in the fields (*NSString**) contained in the XML. These fields will be used as a parameter through an *NSArray.* The IdCloud_DataResult_Delegate protocol will be used to receive the response (once parsed):

- fieldsContentDidComplete Function: This will indicate that parsing has finished and an *NSDictionary* will be obtained with the values and fields, such as a *key*. If no value is returned, you will receive an empty *NSString* (@"").

Parameter type	Name	Parameter definition
NSArray*	_codes	Group of fields in NSString* format to be searched in the xml. You can find the possible values in the "FIELDS" Appendix to the "SDKRoot/Documentation/ IDFast_IDFraud_Result_Fields_EN.pdf" document.
Id<IdCloud_DataResult_Delegate>	_delegate	Delegate that will be used for notification of parsing completion through the fieldsContentDidComplete function.

getResultImages Function:

Through this function a search can be done to find the images (*UIImage**) of the fields (*NSString**) contained in the xml. These fields will be used as a parameter through an *NSArray.* The IdCloud_DataResult_Delegate protocol will be used to receive the response (once parsed):

- imagesContentDidComplete Function: This will indicate that parsing has finished and an *NSDictionary* will be obtained with the values and fields, such as *key*. If no image is found you will receive *[NSNull null]*.

Parameter type	Name	Parameter definition
NSArray*	_codes	Group of fields in NSString* format to be searched in the xml. Possible values can be found in "SDKRoot/Documentation/ IDFast_IDFraud_Result_Fields_EN.pdf" document.
Id<IdCloud_DataResult_Delegate>	_delegate	Delegate that will be used for notification of parsing completion through the fieldsContentDidComplete function.

Example:

In file .h you must convert to the *ViewController* as delegate of the IdCloud_DataResult_Delegate. In this example it is understood that the IdCloud_DataResult data structure has been filled in with information through the processIdCloudWithConf process.

```
@interface ViewController: UIViewController <IdCloud_DataResult_Delegate>
{
    IdCloud_DataResult* m_DataResult;
}
```

The XML can be parsed at any time to search for certain fields.

```
- (IBAction)getFields:(id)sender
{
    NSMutableArray* array = [[NSMutableArray alloc] init];
    [array addObject:@"TYPE"];
    [array addObject:@"SIDE"];
    [array addObject:@"EXPEDITOR"];
    [array addObject:@"NATIONALITY"];
    [array addObject:@"TEST_MRZ_FIELDS_INTEGRITY"];

    [_dataResult getFieldsContent:array andDelegate:self];
}
```

Through the fieldsContentDidComplete function, you can access the parsed values of the getFieldsContent function.

```
- (void) fieldsContentDidComplete:(NSDictionary*)_results
{
    NSString* value = [_results objectForKey: @"TYPE"];
    NSLog(@"%@",value);
}
```

The XML can be parsed at any time to search for certain images.

```
- (IBAction)getImages:(id)sender
{
    NSMutableArray* array = [[NSMutableArray alloc] init];
    [array addObject:@"ImageSignature"];
    [array addObject:@"Image1cut"];
    [array addObject:@"Image2cut"];
    [array addObject:@"ImagePhoto"];

    [_dataResult getResultImages:array andDelegate:self];
}
```

Through the fieldsContentDidComplete function, you can access the values parsed by the getResultImages function.

```
- (void) imagesContentDidComplete:(NSDictionary*)_results
{
    if( [_results objectForKey:@"ImageSignature"] != [NSNull null])
    {
        _img_Content.image = [_results objectForKey:@"ImageSignature"];
    }
    else{
        _img_Content.image = nil;
    }
}
```

The IdCloud_Result data structure will be explained in detail:

Physical location:

IdCloud_Result.h

Declaration:

```
typedef NS_ENUM(NSInteger, IC_Result)
{
    IdCloudResult_OK             = 0,
    IdCloudResult_NOT_PROCESSED  = 1,
    IdCloudResult_NOT_CORRECT    = 2,
    IdCloudResult_ERROR          = 3,
    IdCloudResult_NONE           = 4,
};

typedef NS_ENUM(NSInteger, IC_Error)
{
    IdCloudError_InvalidConfig          = 0,
    IdCloudError_InvalidImages          = 1,
    IdCloudError_InvalidXML             = 2,
    IdCloudError_InvalidJSON            = 3,
    IdCloudError_ConnectionNull         = 4,
    IdCloudError_ServerConnection       = 5,
    IdCloudError_ServerTimeOut          = 6,
    IdCloudError_InvalidIcarLicenseKey  = 7,
    IdCloudError_None                   = 8,
};

@interface IdCloud_Result : NSObject
{}

…(…, readwrite) IC_Error    error;
…(…, readwrite) IC_Result   result;
…(…, readwrite) NSString*   error_desc;
…(…, readwrite) NSString*   error_code;
…(…, readwrite) NSString*   warning;
```

Properties:

error	When the server completes a process correctly, you will receive IdCloudError_None (document has been processed by IdCloud). If not, you may receive: - IdCloudError_InvalidConfig: An invalid IdCloud_Configuration structure has occurred (null or with null values). - IdCloudError_InvalidImages: An invalid IdCloud_LocalImages structure has occurred (null or without any images).

	- IdCloudError_InvalidXML: The XML obtained by the server is corrupt. - IdCloudError_InvalidXML: The JSON obtained by the server is corrupt. - IdCloudError_ConnectionNull: An error occurred when trying to connect to the server. - IdCloudError_ServerConnection: The IDCloud server found an error when processing the document. For more information, consult the error_desc and error_code values described in the "SDKRoot/Documentation/ IDFast_IDFraud_Result_Fields_EN.pdf" document, under the section titled "Security - Methodology of returned errors". - IdCloudError_ServerTimeOut: The maximum wait time for receiving a response from the IDCloud server has elapsed. - IdCloudError_InvalidIcarLicenseKey: invalid Icar License Key.
result	You may receive the following results: - IdCloudResult_ERROR: An error has occurred; consult the "error" variable described above. - IdCloudResult_OK: The document(s) were validated correctly. - IdCloudResult_NOT_CORRECT: The document(s) were not validated correctly (message appears in "warning" field). - IdCloudResult_NOT_PROCESSED: The document could not be processed (message in the "warning" field).
error_desc	If an IdCloudError_ServerConnection error occurs, the server will provide more information in this field. Consult the "IDCloud Manual de integración.pdf" document, under the section titled "Methodology of returned errors".
error_code	If an IdCloudError_ServerConnection error occurs, the server will provide more information in this field. Consult the "SDKRoot/Documentation/IDFast_IDFraud_Result_Fields_EN.pdf" document, under the section titled "Methodology of returned errors".
warning	If you receive error=IdCloudError_None and result=IdCloudResult_NOT_PROCESSED or IdCloudResult_NOT_CORRECT, more information can be obtained from the server in this field. For more information, consult section 1.1.2 of the "IDCloud Integration Manual.pdf" document.

3.4. CONFIRM SERVER CONNECTION CREDENTIALS

Before connecting to the IDCloud server in order to process documents, in your developing app version you can confirm that the parameters of the IdCloud_Configuration structure are valid. To do this, execute the checkIdCloudWithConf function of the IdCloud_Connection class.

To receive a response regarding the validation of credentials, use the same IdCloud_Connection_Delegate protocol as was used to process documents. That is, implement the connectionDidCancel and connectionDidComplete functions in order to find out if the validation was completed correctly.

Important: this function is only to check your credentials in your developing version, it's not for your final app version.

Example:

In file .h you must convert to the VC as delegate of the IdCloud_Connection_Delegate. It is important to keep the life cycle of the IdCloud_Connection instance at the same level as its containing class.

```
@interface ViewController: UIViewController <IdCloud_Connection_Delegate>
{
   IdCloud_Connection*  m_IdCloud_Connection;
}
```

The following function shows the complete code to validate credentials.

```
- (void) checkConnection
{
    IdCloud_Configuration*  idCloud_Conf;
    idCloud_Conf = [[IdCloud_Configuration alloc] init];
    idCloud_Conf.user           = @"user";
    idCloud_Conf.password       = @"1234567";
    idCloud_Conf.company        = @"icar";
    idCloud_Conf.server_Port    = @"50061";
    idCloud_Conf.server_Timeout = 20000; //in milliseconds
    idCloud_Conf.server_URL     = @"https://demoidcloud.icarvision.com";
    idCloud_Conf.icarLicenseKey = @"HUiedjeioj...";

    m_IdCloud_Connection = [[IdCloud_Connection alloc] init];
    [_idCloud_Connection checkIdCloudWithConf:idCloud_Conf delegate:self];
}
```

The connectionDidComplete function will tell you if the validation was successful.

```
- (void) connectionDidComplete
{
    NSLog(@"Check Ok");
}
```

The connectionDidCancel function will tell you if the validation was unsuccessful.

```
- (void) connectionDidCancel:(IdCloud_Result*)_result
{
    switch (_result.error)
    {
        case IdCloudError_ConnectionNull:
            NSLog(@"Error: Connection Null");
            break;
        case IdCloudError_InvalidConfig:
            NSLog(@"Error: IdCloudError_InvalidConfig");
            break;
        case IdCloudError_InvalidImages:
            NSLog(@"Error: IdCloudError_InvalidImages");
            break;
        case IdCloudError_ServerConnection:
            NSLog(@"Error: IdCloudError_ServerConnection");
            NSLog(@"connectionDidCancel_desc: %@",_result.desc);
            break;
        default:
            break;
    }
}
```

4. OFFLINE PROCESSING MODULE

4.1. ICARCAPTURE_VC FOR OFFLINE PROCESSING

As explained in section "2. IMAGE CAPTURE MODULE", through the IcarCapture_VC, IcarCapture_NC, IcarCapture_Configuration, and IdCloud_LocalImages classes, you can get an image from the phone to send to the IDCloud server, since the interfaces of these classes also allow certain processes to be carried out offline. For this of the SDK you can use local processing to automatically obtain:

- MRZ lines of the document (and parsing)
- Contents of the PDF417 bar code (and parsing depending on the manual country configuration).

In later versions of the SDK, more offline processes will be enabled.

Before showing an example of how to execute a certain process offline, the exclusive IcarCapture_Configuration and IdCloud_LocalImages interface is explained in detail for this section:

Physical location:

IcarCapture_Configuration.h

Declaration:

```
...
…(…, readwrite)     bool enableAutoFocusOnTouch;     //default: true
…(…, readwrite)     int mrzNumHelpView      //default: -1. (0: never, -1: infinite)
…(…, readwrite)     float mrzTimerHelpView;    //defualt 3f seconds
…(…, readwrite)     NSString* icarLicenseKey;
…(…, readwrite)     NSString* pdf417LicenseKey;
```

Properties:

enableAutoFocusOnTouch	Enable auto-focus upon pressing the screen in IcarCapture_VC.
mrzNumHelpView	Activate the help view in the IcarCapture_VC screen (see image 2). The values may be: • 0: The help window will never be shown. • -1: It will always be shown. • X > 0: It will only be shown x times.
mrzTimerHelpView	Time during the help screen will be shown for IcarCapture_VC (see Illustration 4: IcarCapture_VC-MRZ with help view.).
icarLicenseKey	Important: All acquisitions require a valid Icar license.

pdf417LicenseKey	License key for pdf417 capture.

Illustration 4: IcarCapture_VC-MRZ with help view.

Physical location:

IdCloud_ LocalImages.h

Declaration:

```
@interface IdCloud_LocalImages : NSObject{}
...
…(…, readwrite)     IcarParsedMrz*     parsedMRZ;
…(…, readwrite)     IcarParsedPdf417* parsedPDF417;

@interface IcarParsedMrz: NSObject
{}

@property (nonatomic, copy) NSString*   mrzLines;
@property (nonatomic, copy) NSString*   globalIntegrity;
@property (nonatomic, copy) NSString*   fieldsIntegrity;
@property (nonatomic, copy) NSString*   fields;

@end

@interface IcarParsedPdf417: NSObject
{}

@property (nonatomic, copy) NSData*     allNSData;
@property (nonatomic, copy) NSString*   allStringData;

@property (nonatomic, copy) NSString*   custom_1;
@property (nonatomic, copy) NSString*   custom_2;
@property (nonatomic, copy) NSString*   custom_3;
@property (nonatomic, copy) NSString*   docnumber;
@property (nonatomic, copy) NSString*   surname;
@property (nonatomic, copy) NSString*   surname2;
@property (nonatomic, copy) NSString*   name;
@property (nonatomic, copy) NSString*   name2;
@property (nonatomic, copy) NSString*   sex;
@property (nonatomic, copy) NSString*   birthdate;
@property (nonatomic, copy) NSString*   custom_4;
@property (nonatomic, copy) NSString*   blood_type;
@end
```

Properties:

parsedMRZ.mrzLines	MRZ lines of the processed document.
parsedMRZ.globalIntergity	Global integrity of the processed MRZ code.
parsedMRZ.fieldsIntegrity	Integrity of each field of MRZ code processed.
parsedMRZ.fields	The MRZ code fields processed.

parsedPDF417.allNSData	Complete PDF417 information processed in NSData* format.
parsedPDF417.allStringData	Complete PDF417 information processed in NSString* format.
parsedPDF417.custom_1 ... parsedPDF417.blood_type	Specific fields of the NSString* format of the PDF417 processed.

Example:

In file .h you must convert to the *ViewController* as delegate of the IcarCapture_Delegate. It is important to keep the life cycle of the IcarCapture_Configuration and IdCloud_LocalImages instances at the same level as their containing class.

```
@interface IdCloud_VC : UIViewController <IcarCapture_Delegate>
{
    IdCloud_LocalImages*        m_IdCloud_LocalImages;
    IcarCapture_Configuration*  m_IcarCapture_Config;
}
```

Start up the structures. To carry out the PDF417 code process, you simply have to modify the code "`[m_IcarCapture_Config setInternalProcess: CaptureProcess_CAPTURE_PDF417];`"

```
- (void)viewDidLoad
{
   [super viewDidLoad];
   m_IdCloud_LocalImages = [[IdCloud_LocalImages alloc] init];
   m_IcarCapture_Config  = [[IcarCapture_Configuration alloc] init];
   m_IcarCapture_Config.icarLicenseKey = @"Hiuojdkle....";
   [m_IcarCapture_Config setInternalProcess: CaptureProcess_CAPTURE_MRZ];
}
```

The following code shows the execution of the image capture module..

```
- (IBAction)showImageCapture:(id)sender
{
    IcarCapture_VC*    vc    =    [[IcarCapture_VC    alloc]    initWithDelegate:self
andConfiguration:m_IcarCapture_Config            andLocalImage:m_IdCloud_LocalImages
andPosImage:0];

    IcarCapture_NC *nav = [[IcarCapture_NC alloc] init];
    [nav setViewControllers:[NSArray arrayWithObjects:vc, nil]];
    [self presentViewController:nav animated:NO completion:nil];
}
```

Through the *captureDidCancel* and *captureDidComplete* functions, you can receive notifications regarding the final capture result. It is important to execute *dismissViewControllerAnimated* in these functions to make the IcarCapture_VC disappear.

```
#pragma mark - IcarCapture_Delegate
- (void) captureDidCancel
{
    if (_result == CaptureResult_CANCELED_BY_USER)
    {
        [self dismissViewControllerAnimated:NO completion:nil];
    }
    else if (_result == CaptureResult_ERROR)
    {
        [self dismissViewControllerAnimated:NO completion:nil];
    }
    else if (_result == CaptureResult_NO_CREDIT)
    {
        NSLog(@"MainDemo_TVC: ICAR_CaptureResult = CaptureResult_NO_CREDIT");
    }
}
- (void) captureDidComplete
{
    [self dismissViewControllerAnimated:NO completion:nil];

    m_bMRZ_Scanner = false;
    UIAlertView *alert = [[UIAlertView alloc]
              initWithTitle:@"MRZ:"
              message:m_IdCloud_LocalImages.parºsedMRZ.mrzLines
              delegate:self
              cancelButtonTitle: nil
              otherButtonTitles:@"Continue", nil];

    [alert show];
}
```

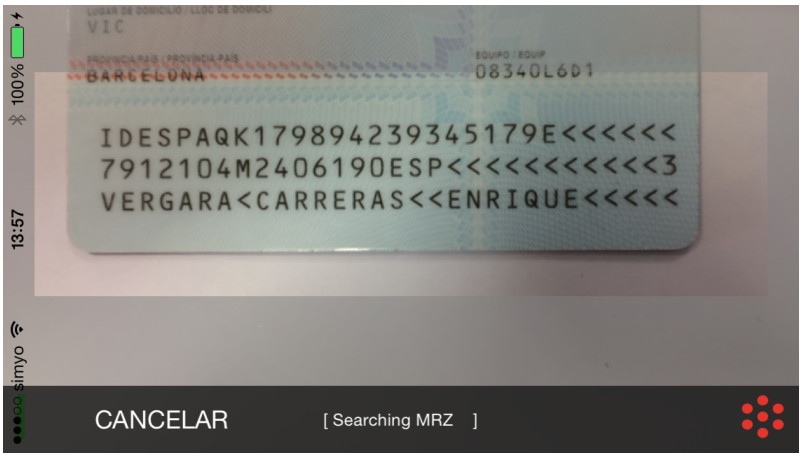

Illustration 5: IcarCapture_VC screen capture for MRZ processing.

Illustration 6: IcarCapture_VC screen capture for PDF417 processing.

4.2. ICAR_PAYMENT SERVER

In order to enable the execution of certain processes in offline mode (i.e. to carry out all calculations locally on the phone) it is necessary for the device to be previously connected to an Icar server. This connection, understood as a purchase, will record the number of times the phone can execute a certain process on the device.
As an added functionality, a connection to the server is provided in order to return unused processes, e.g. before retiring a mobile device or reinstalling the application. These are situations in which you would lose the number of processes that had not yet occurred if this function was not used.
The server connection is done in the Offline Process module, i.e. using a group of classes whose interface is explained below:

Important: All purchase connections to the Icar server use a client certificate that is valid during 2 years from the date of the version release. After that date, no more purchase connections can be donned with this version of the SDK..

Important: In version 1.5.18 and later all SDK integrations require a valid Icar license. The license information can be specified using the "icarLicenseKey" field (for more information contact the sales team of Icar).

4.3. CONFIGURATION FOR THE ICAR_PAYMENT SERVER CONNECTION.

First, the necessary information should be entered to connect to the Icar server. The IcarPayment_Configuration class is available for this.

Physical location:

IcarPayment_Configuration.h

Declaration:

```
@interface IcarPayment_Configuration : NSObject{}
…(…, readwrite) NSString*              user;
```

```
…(…, readwrite) NSString*                password;
…(…, readwrite) NSString*                company;
…(…, readwrite) ICAR_CaptureProcess      process;
…(…, readwrite) int                      ammount;
…(…, readwrite) int                      server_Timeout;
…(…, readwrite) NSString*                icarLicenseKey;
```

Properties:

user	Credential with user information used to identify oneself to the Icar server.
password	Credential with password information used to identify oneself to the Icar server.
company	Credential with the company information used to identify oneself to the Icar server.
process	Type of process to purchase. The type of process (enumerate ICAR_CaptureProcess) to purchase may be: • CaptureProcess_CAPTURE_MRZ • CaptureProcess_CAPTURE_PDF417
ammount	Quantity of processes to record as purchased.
server_Timeout	Maximum wait time without receiving a response from the Icar server. Expressed in milliseconds. By default its value is 2 seconds.
icarLicenseKey	Important: All payments require a valid Icar license.

Example:

```
m_Payment_Configuration  = [[IcarPayment_Configuration alloc] init];

m_Payment_Configuration.user             = @"user";
m_Payment_Configuration.password         = @"pwd123";
m_Payment_Configuration.company          = @"companyID";
m_Payment_Configuration.process          = CaptureProcess_CAPTURE_MRZ;
m_Payment_Configuration.ammount          = 50;
m_Payment_Configuration.server_Timeout   = 20000; //20 segundos
m_Payment_Configuration.icarLicenseKey   = @"HUiehjeh...";
```

4.4. CONNECTION TO ICAR_PAYMENT SERVER TO MAKE THE PURCHASE

To connect to the Icar server, you must use the IcarPayment_Manager class.

Physical location:
IcarPayment_Manager.h

Declaration:

```
#pragma mark - Delegates and notifications
//------ICAR PAYMENT DELEGATE---------
@protocol IcarPayment_Delegate <NSObject>
@required
- (void) paymentDidCancel:(IcarPayment_Result*)_result;
- (void) paymentDidComplete:(IcarPayment_Result*)_result;
@end


//-------ICAT PAYMENT MANAGER----------
```

```
@interface IcarPayment_Manager : NSObject
{}


- (int) getProcessCounter:(ICAR_CaptureProcess) _type;

- (void) buyProcess:(IcarPayment_Configuration*) _conf
       andDelegate:(id<IcarPayment_Delegate>)_delegate;

- (void) retryBuyProcess:(id<IcarPayment_Delegate>)_delegate;


- (void) returnProcesses:(IcarPayment_Configuration*) _conf
            andDelegate:(id<IcarPayment_Delegate>)_delegate;

- (void) retryReturnProcesses:(id<IcarPayment_Delegate>)_delegate;
```

getProcessCounter Function:

This function determines the number of remaining processes to be carried out on the device. If the number is recorded locally on the device, it is not necessary to connect to a server. The function can be carried out for a specific process type.

Parameter type	Name	Parameter definition
ICAR_CaptureProcess	_type	Type of process to execute: • CaptureProcess_CAPTURA_MRZ • CaptureProcess_CAPTURA_PDF417

buyProcess Function:

This function allows you to connect to the Icar server and obtain more processes of a given type. To do this, you must introduce a configuration with the process type and quantity to be purchased, in addition to designating a delegate responsible for managing these events.

- paymentDidCancel Function: This will communicate the cancellation of a purchase. You can check the error type using parameter _result of the IcarPayment_Result; the IcarPayment_Result class is described in section 4.3. CONFIGURATION FOR THE ICAR_PAYMENT SERVER CONNECTION."

  Important: if you receive a connection error, the library integrator must retry the purchase process using the retryBuyProcess function until it has been done successfully. This is necessary in order to prevent the server from computing the purchase and the terminal not updating the number of locally recorded processes. If a new purchase or return process is carried out before the connection error is resolved, the retry cannot be done.

- paymentDidComplete Function: This will notify you that the purchase was successful. It will automatically update the number of processes recorded on the mobile device.

Parameter type	Name	Parameter definition
IcarPayment_Configuration *	_conf	Configuration of the purchase to be made. The contents of the "amount" field should be a number greater than 0; otherwise you will receive an error in the form of a paymentDidCancel event.
Id<IcarPayment_Delegate>	_delegate	Delegate that will be used for notification of purchase completion.

retryBuyProcess Function:

As previously stated, in order to prevent the server from computing the purchase carried out through the buyProcess function and the terminal not recording it, the library integrator must use this function upon receiving the paymentDidCancel event. As entry parameters, you must specify the delegate in charge of managing purchase events and the context; no other parameters are necessary since the configuration of the purchase has already been saved in the memory.

Parameter type	Name	Parameter definition
Id<IcarPayment_Delegate>	_delegate	Delegate which will be used for notification of completion of purchase.

returnProcesses Function:

Use this function to return the processes purchased in the terminal but not yet used. To do this, you must specify the types of processes to be returned using the parameter _conf and the delegate responsible for managing such events:

- paymentDidComplete Function: This will communicate to the server that the process return has been carried out successfully. Additionally, the terminal will update the value of locally registered processes to 0.
- paymentDidCancel Function: Notifies user of any errors during the return process. You can find the error type with parameter _result.

Important: if you receive a connection error, the library integrator must retry the return process using the retryReturnProcesses function. If a new purchase or return process is carried out before the connection error is resolved, the retry cannot be done.

Parameter type	Name	Parameter definition
IcarPayment_Configuration*	_conf	Configuration of the purchase to be made. You only have to specify the credentials and the type of process to return. The number of processes to be returned is obtained automatically from the number recorded in the terminal.
Id<IcarPayment_Delegate>	_delegate	Delegate which will be used to notify user of completion of the process return.

retryReturnProcesses Function:

If you receive paymentDidCancel when returning processes, the server may not

compute the return of processes but the terminal may still update the number of processes recorded locally to 0. In this case the library integrator must use the retryReturnProcesses function. As entry parameters you must specify the delegate in charge of managing return events; no other parameters are necessary since the configuration of the return has already been saved in the permanent memory.

Parameter type	Name	Parameter definition
Id<IcarPayment_Delegate>	_delegate	Delegate which will be used to notify user of completion of the process return.

Example:

In file .h you must convert to the *ViewController* as delegate of the IcarCapture_Delegate.

```
@interface ViewController: UIViewController <IdCloud_Connection_Delegate>
{
    IcarPayment_Manager*         m_IcarPayment_Manager;
    IcarPayment_Configuration    m_IcarPayment_Configuration;
}
```

It is important to keep the life cycle of the IcarPayment_Manager instance at the same level as its containing class.

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    m_IcarPayment_Manager = [[IcarPayment_Manager alloc] init];
    m_IcarPayment_Configuration = [[IcarPayment_Configuration alloc] init];
}
```

The following function shows the full code for making a purchase.

```
- (void) buyMRZProcess
{
  m_Payment_Configuration.user      = @"user";
  m_Payment_Configuration.password = @"pwd123";
  m_Payment_Configuration.company   = @"companyID";

  m_Payment_Configuration.process = CaptureProcess_CAPTURE_MRZ;
  m_Payment_Configuration.ammount = 50;
  m_Payment_Configuration.server_Timeout = 20000;
  m_Payment_Configuration.icarLicenseKey = @"JIOjdksd...";

  [m_Payment_Manager buyProcess: m_Payment_Configuration andDelegate:self];
}
```

Finally, the paymentDidComplete and paymentDidCancel functions are implemented in order to manage the delegate's events.

```
#pragma IcarPayment_Delegate
- (void) paymentDidComplete:(IcarPayment_Result*)_result
{
  int counter_MRZ = [paymentManager getProcessCounter:CaptureProcess_CAPTURE_MRZ];
   NSLog(@"MRZ:%d",counter_MRZ);
}
- (void) paymentDidCancel:(IcarPayment_Result*)_result
{
  //For example, retry the purchase process...
  [m_Payment_Manager retryBuyProcess:self];
}
```

4.5. PURCHASE PROCESS RESULT

Upon finishing a purchase or return of processes, an instance of the IcarPayment_Result class will be obtained using the IcarPayment_Delegate protocol. This IcarPayment_Result class is detailed below:

Physical location:

IcarPayment_Result.h

Declaration:

```
typedef NS_ENUM(NSInteger, IP_Result)
{
    PaymentResult_BOUGHT        = 0,
    PaymentResult_ERROR         = 1,
    PaymentResult_NONE          = 2,
};

typedef NS_ENUM(NSInteger, IP_Error)
{
    PaymentError_InvalidConfig          = 0,
    PaymentError_InvalidAmmount         = 1,
    PaymentError_ConnectionNull         = 2,
    PaymentError_ServerConnection       = 3,
    PaymentError_LastPurchasedCompleted = 4,
    PaymentError_LastReturnCompleted    = 5,
    PaymentError_InvalidIcarLicenseKey  = 6,
    PaymentError_None                   = 7,
};


//------ICAR PAYMENT RESULT----------------
@interface IcarPayment_Result : NSObject
{}

…(…, readwrite) IP_Result result;
…(…, readwrite) IP_Error  error;
…(…, readwrite) NSString* error_desc;
```

Properties:

error	When the server correctly completes a purchase or return of processes, you will receive PaymentError_NONE. If not, you may receive: • PaymentError_InvalidConfig: An invalid IcarPayment_Configuration structure has occurred (null or with null values). • PaymentError_InvalidAmmount: In order to make a purchase, the value of the "amount" field in the IcarPayment_Configuration should be greater than 0. Otherwise, you may receive this error. • PaymentError_ConnectionNull: An error occurred when trying to connect to the server. • PaymentError_ServerConnection: The Icar server encountered an error when processing the purchase order or return of processes. For more information, consult the error_desc value. • PaymentError_LastPurchasedCompleted: This error will appear if you successfully make a purchase and implement the retryBuy function. • PaymentError_LastReutrnCompleted: This error will appear if you

	successfully make a return and implement the retryReturnProcesses Function. • PaymentError_InvalidIcarLicenseKey: error with invalid icarLicenseKey.
result	You may receive the following results: • PaymentResult_ERROR: Errors occurred during the purchase or return process; consult the above mentioned “error” variable. • PaymentResult_BOUGHT: The purchase or return process was successful. • PaymentResult_NONE: An unknown error occurred when executing functions in the purchase or return process.
error_desc	If a PaymentError_ServerConnection error occurs, the server will provide more information in this field. • 1_ERROR_SERVER: Internal Icar payment server error. • 2_ERROR_INCORRECT_USER: Credential error (user, company, password). • 3_ERROR_INCORRECT_PROCESS_NAME: Process name error. • 4_ERROR_PURCHASE_LIMITATION: When using test credentials and making a purchase, you may receive a message stating that you have exceeded the maximum number of purchase processes allowed, in which case this error will appear. • 5_ERROR_USER_DISABLE: The user has been disabled by Icar.

5. QUICK REFERENCE GUIDE

In SDK you can find two demo projects to demonstrate the use of the three modules:

- If you want to see a simpler sdk example, that show how to capture front and back from doc and selfie to be sent to the idcloud server, you could see section "5.1. EXAMPLE_CAPTURE FRONT/BACK/FACE AND SEND TO ICAR CLOUD SERVER".
- In case you want to use only the offline process mrz or pdf417, you can see the section "5.2. EXAMPLE_OFFLINE_PROCESSES".

In section "5.3. CUSTOMIZE AUTOFOTO PROCESS" you can see how to customize the layout of certain processes. Finally in section "5.4. SENDING CAPTURED IMAGES WITHOUT USING IDLCOUD CONNECTION" are indicated the steps to capture the documentation (and selfie) and send them to its own server.

5.1. EXAMPLE_CAPTURE FRONT/BACK/FACE AND SEND TO ICAR CLOUD SERVER

Physical location:
SDKRoot / Demos SDK / CaptureAndSend

Imagine the next flow of a hypothetical application:

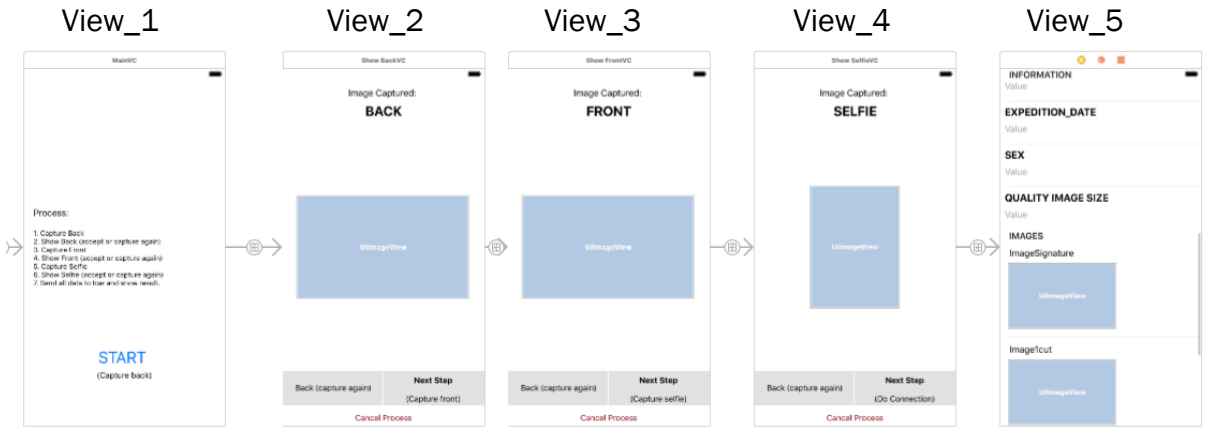

Illustration 7: Hypothetical storyboard of application.

- **View_1**: Capture Back by pressing Start, on captureDidComplete go to View_2.
- **View_2**: Show Back in viewdidload, capture Front by pressing Next. On captureDidComplete go to View_3.
- **View_3**: Show Front in viewdidload, capture Selfie by pressing Next. On captureDidComplete go to View_4.

- **View_4**: Show Selfie in viewdidload, do IDCloud Server connection by pressing Next. On connectionDidComplete go to View_5.
- **View_5**: Show results.

View_1:

Start_Clicked: localImages is created in this view and is propagated by the different views. This instance stores the different captured images, so it cannot be deleted. In this case we use slot 0 of localImage to store front image.

```
//Capture FRONT DOCUMENT:
localImages = [[IdCloud_LocalImages alloc] init]; //alloc-->Quick way to reset the structure
in charge of storing the captured images
captureConfig  = [[IcarCapture_Configuration alloc] init];
captureConfig.enableVibrationAutoFoto = YES;
captureConfig.captureDocWithAutoFoto = YES;
[captureConfig setInternalProcess:CaptureProcess_CAPTURE_IMAGE];
captureConfig.icarLicenseKey = DemoSDK_icarLicenseKey;
captureConfig.customized_AutoFoto.text_Toolbar = @"[Place the ID (BACK) within the Box ]";

IcarCapture_VC* vc = [[IcarCapture_VC alloc] initWithDelegate:self
                                                andConfiguration:captureConfig
                                                   andLocalImage:localImages
                                                     andPosImage:0];

IcarCapture_NC *nav = [[IcarCapture_NC alloc] init];
[nav setViewControllers:[NSArray arrayWithObjects:vc, nil]];
[self presentViewController:nav animated:NO completion:nil];
```

CapturedDidComplete: Execute segue to ShowBack_VC (View_2), and pass the info localImages (the instance with de back image captures) and captureConfig.

```
- (void) captureDidComplete
{
    //It's very important once we receive the image from the IcarCapture_VC to call the
funcion dismissViewControllerAnimated
    [self dismissViewControllerAnimated:NO completion:^
    {
        [self performSegueWithIdentifier:@"FromStart2ShowBack" sender:nil];
    }];
}


 #pragma mark - Navigation
// In a storyboard-based application, you will often want to do a little preparation before
navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
    if ([[segue identifier] isEqualToString:@"FromStart2ShowBack"])
    {
        ShowBack_VC* _vc = [segue destinationViewController];
        _vc.localImages     = localImages;
        _vc.captureConfig   = captureConfig;
    }
}
```

View_2:

viewDidLoad: show the back image captured in previous view.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    _back_img.image = [_localImages getImage:0];
}
```

Next_Clicked: We use localImages to store the front image on slot 1.

```
- (IBAction)next_Clicked:(id)sender
{
    //Capture BACK DOCUMENT:
    [_captureConfig setInternalProcess:CaptureProcess_CAPTURE_IMAGE];
    _captureConfig.icarLicenseKey = DemoSDK_icarLicenseKey;
    _captureConfig.enableVibrationAutoFoto = YES;
    _captureConfig.captureDocWithAutoFoto = YES;
_captureConfig.customized_AutoFoto.text_Toolbar = @"[ Place the ID (FRONT) within the Box ]";

    IcarCapture_VC* vc = [[IcarCapture_VC alloc] initWithDelegate:self
                                                    andConfiguration:_captureConfig
                                                       andLocalImage:_localImages
                                                         andPosImage:1];

    IcarCapture_NC *nav = [[IcarCapture_NC alloc] init];
    [nav setViewControllers:[NSArray arrayWithObjects:vc, nil]];
    [self presentViewController:nav animated:NO completion:nil];
}
```

CapturedDidComplete: Execute segue to ShowFront_VC (View_3), and pass the info localImages (the instance with de back and front images captured) and captureConfig.

```
- (void) captureDidComplete
{
    //It's very important once we receive the image from the IcarCapture_VC to call the
funcion dismissViewControllerAnimated
    [self dismissViewControllerAnimated:NO completion:^
     {
         [self performSegueWithIdentifier:@"FromShowBack2ShowFront" sender:nil];
     }];
}

#pragma mark - Navigation
// In a storyboard-based application, you will often want to do a little preparation before
navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString:@"FromShowBack2ShowFront"])
    {
        ShowFront_VC* _vc = [segue destinationViewController];
        _vc.localImages     = _localImages;
        _vc.captureConfig   = _captureConfig;
    }
}
```

View_3:

viewDidLoad: show the front image captured in previous view.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    _front_img.image = [_localImages getImage:1];
}
```

Next_Clicked: We use localImages to store the selfie on _localImages.faceDetected.

```
- (IBAction)next_Clicked:(id)sender
{
    //Capture FACE:
    [_captureConfig setInternalProcess:CaptureProcess_CAPTURE_FACE_DETECTION];
    _captureConfig.icarLicenseKey = DemoSDK_icarLicenseKey;
    _captureConfig.enableVibrationFaceDetection = YES;
    _captureConfig.liveness = YES;
    IcarCapture_VC* vc = [[IcarCapture_VC alloc] initWithDelegate:self
                                                    andConfiguration:_captureConfig
                                                       andLocalImage:_localImages
                                                         andPosImage:-1];

    IcarCapture_NC *nav = [[IcarCapture_NC alloc] init];
    [nav setViewControllers:[NSArray arrayWithObjects:vc, nil]];
```

```
    [self presentViewController:nav animated:NO completion:nil];
}
```

CapturedDidComplete: Execute segue to ShowSelfie_VC (View_4), and pass the info localImages (the instance with de back, front and selfie images captured) and captureConfig.

```
- (void) captureDidComplete
{
    //It's very important once we receive the image from the IcarCapture_VC to call the
funcion dismissViewControllerAnimated
    [self dismissViewControllerAnimated:NO completion:^
     {
         [self performSegueWithIdentifier:@"FromShowFront2ShowSelfie" sender:nil];
     }];
}

#pragma mark - Navigation
// In a storyboard-based application, you will often want to do a little preparation before
navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString:@"FromShowFront2ShowSelfie"])
    {
        ShowSelfie_VC* _vc = [segue destinationViewController];
        _vc.localImages     = _localImages;
        _vc.captureConfig   = _captureConfig;
    }
}
```

View_4:

viewDidLoad: show the selfie image captured in previous view.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    _selfie_img.image = _localImages.faceDetected;
}
```

Next_Clicked: We use localImages to send front and back images to IDCloud Server. Previsously we use idCloud_Config to set the selfie image with the correct resize.

```
IdCloud_Configuration*  idCloud_Conf;
    idCloud_Conf = [[IdCloud_Configuration alloc] init];

    idCloud_Conf.user           = DemoSDK_IDCloud_User;
    idCloud_Conf.password       = DemoSDK_IDCloud_Password;
    idCloud_Conf.company        = DemoSDK_IDCloud_Company;
    idCloud_Conf.server_Port    = @"50061";
    idCloud_Conf.server_Timeout = 20000; //in milliseconds
    idCloud_Conf.server_URL     = @"https://demoidcloud.icarvision.com";
    idCloud_Conf.icarLicenseKey = DemoSDK_icarLicenseKey;;

    //Configure the images to be sent to the server:


    [idCloud_Conf setFaceValidation:_localImages.faceDetected];
    //Execute the process in the server:
    dataResult = [[IdCloud_DataResult alloc] init];

    IdCloud_Connection* m_IdCloud_Connection = [[IdCloud_Connection alloc] init];
    [m_IdCloud_Connection processIdCloudWithConf:idCloud_Conf
                                          images:_localImages
                                        delegate:self
                                   andDataResult:dataResult];
```

ConnectionidComplete: Finally on this function we execute segue to IcarCloudResults_TVC (View_5), and pass the info localImages.

```
- (void) connectionDidComplete:(IdCloud_Result*)_result
{
```

```
    idCloud_Result = _result;
    [alert dismissViewControllerAnimated:YES completion:nil];
    [self performSegueWithIdentifier:@"FromShowSelfie2ShowCloudResults" sender:nil];
}

#pragma mark - Navigation
// In a storyboard-based application, you will often want to do a little preparation before
navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString:@"FromShowSelfie2ShowCloudResults"])
    {
        IcarCloudResults_TVC* _vc = [segue destinationViewController];
        _vc.idCloud_DataResult = dataResult;
        _vc.idCloud_Result = idCloud_Result;
        _vc.localImages = _localImages;
    }
}
```

5.2. EXAMPLE_OFFLINE_PROCESSES

Physical location:
SDKRoot / Demos SDK / OfflineProcesses

In this example, the basic actions are shown to be able to parse a mrz or pdf417 locally in the mobile. Before executing any of these local processes, it is necessary that the integrator performs a pre-purchase of a certain amount of processes (button ). In case you want to return the unprocessed clicks, you can use a call from the sdk (button ).
In order to perform the purchase or return of cliks, it is necessary to configure correctly the credentials used in these two processes (the credentials to validate against the icar payment server). The example shows the credentials used, as well as a button to modify them at runtime.

5.3. CUSTOMIZE AUTOFOTO PROCESS

Using the following screenshot as a reference, you can customize the layout of auto-foto process with:

Illustration 8: Description of the customizable parts in the autofoto process.

There is a new structure called Customized_AutoFoto to customize the layout of autofoto process:

Physical location:

IcarCapture_Configuration.h

Declaration:

```
@interface Customized_AutoFoto : NSObject
{}

//---Properties:
…(…, readwrite)  UIColor*    color_BackgroundLayout;
…(…, readwrite)  float       alpha_BackgroundLayout;     //[0.0f - 1.0f]
…(…, readwrite)  UIColor*    color_Toolbar;              //Color and alpha.
…(…, readwrite)  float       alpha_Toolbar;              //[0.0f - 1.0f]
…(…, readwrite)  NSString*   text_Toolbar;
…(…, readwrite)  bool        show_HelpButton;
…(…, readwrite)  NSString*   resource_Template;
…(…, readwrite)  NSString*   resource_btnCancel;
…(…, readwrite)  UIColor*    color_DetectedFrames;
…(…, readwrite)  UIColor*    color_NotDetectedFrames;
…(…, readwrite)  int         time_ToDisappear_Template; //in milliseconds.

@interface IcarCapture_Configuration : NSObject
{}
…(…, readwrite)  Customized_AutoFoto*    customized_AutoFoto;
```

Example of a possible customization:

```
IcarCapture_Configuration*  m_IcarCapture_Config;

- (void) customizeIcarLayouts
```

```
{
    //1
    m_IcarCapture_Config.customized_AutoFoto.color_BackgroundLayout = [self
colorWithHexString:@"#0000A6DD"];
    m_IcarCapture_Config.customized_AutoFoto.alpha_BackgroundLayout = 0.7f;

    //2
    m_IcarCapture_Config.customized_AutoFoto.show_HelpButton = NO;

    //5
    m_IcarCapture_Config.customized_AutoFoto.text_Toolbar = @"Posicione o documento dentro do molde";

    //4
    m_IcarCapture_Config.customized_AutoFoto.color_Toolbar = [self colorWithHexString:@"#0055A6DD"];
    m_IcarCapture_Config.customized_AutoFoto.alpha_Toolbar = 1.0f;

    //3
    m_IcarCapture_Config.customized_AutoFoto.resource_Template = @"mask_cnh.png";
    m_IcarCapture_Config.customized_AutoFoto.time_ToDisappear_Template = -1;// in milliseconds (-1 means
it will never disappear)

    //You can change the graphic resource of close button (now is a cross)
    m_IcarCapture_Config.customized_AutoFoto.resource_btnCancel = @"button_back.png";

}
```

5.4. SENDING CAPTURED IMAGES WITHOUT USING IDLCOUD CONNECTION

In case you want to use the icar capture module, to get the front and/or back of a document, and send all this information to your server without using IdcloudConnection, it is very important that you make use of the getDeviceInfo function of the Class IdCloud_LocalImages. This function returns a string with important information necessary for the correct reading of the document by the icar server.

Imagine that you have captured the front and back of a document, as well as the selfie. In this case you should obtain the following information:

(Note: we assume that the _localImages instance of the IdCloud_LocalImages class is the one you used to call the icar capture with IcarCapture_VC).

```
- (void)SendToMyServerAllData {
NSString* img1_base64 = [_localImages getImage_base64:0];
NSString* img2_base64 = [_localImages getImage_base64:1];
IdCloud_Configuration*  idCloud_Conf;
idCloud_Conf = [[IdCloud_Configuration alloc] init];
[idCloud_Conf setFaceValidation:_localImages.faceDetected];
NSString* deviceInfo_XML_0 = [_localImages getDeviceInfo:0 withConf:idCloud_Conf];
NSString* deviceInfo_XML_1 = [_localImages getDeviceInfo:1 withConf:idCloud_Conf];
//TODO.. send all this NSString to your server...
}
```

6. APPENDIX

6.1. TABLE WITH LITERALS TO BE TRANSLATED

The following table lists are the literals that must be added into *Localizable.strings* of the application (for each language).

Literal Identifier	Description
icar_sdk_placeIdWithinTheBox	Place the ID within the Box
icar_sdk_notEnoughLight	Not enough light...
icar_sdk_searching_mrz	Searching MRZ
icar_sdk_detected_mrz	MRZ found!
icar_sdk_saving_photo_capture	Saving photo capture . . .
icar_sdk_btn_capture	Capture
icar_sdk_btn_cancel	Cancel
icar_sdk_btn_ok	Ok
icar_sdk_stay_still	Stay still, the image being captured
icar_sdk_placeDocInsideBox	Place the document inside the box
icar_sdk_please_come_closer	PLEASE, COME CLOSER
icar_sdk_ok_stay_still	OK!, STAY STILL
icar_sdk_place_face_inside	PLACE THE FACE INSIDE
icar_sdk_picture_token	PICTURE TOKEN
icar_sdk_lbl_no_credits	ICAR_SDK_ERROR: No credits, please contact with Icar Vision
icar_sdk_go_back	PLEASE, GO BACK
icar_sdk_blink_eyes	PLEASE, BLINK EYES
icar_sdk_accept_thanks	ACCEPTED THANK YOU
icar_sdk_turn_head_left_right	TURN YOUR HEAD LEFT AND RIGHT
icar_sdk_please_look_front	PLEASE LOOK FRONT
icar_sdk_turn_head_oher_side	TURN YOUR HEAD THE OTHER SIDE
icar_sdk_please_be_serious	PLEASE BE SERIOUS
icar_sdk_big_smile	BIG SMILE! :-)
icar_sdk_yes	YES
icar_sdk_not	NOT

In the DemoSDK application called CaptureAndSend you can find an example of Localizable.strings with the translations of the literals into the Catalan language.

The path where to find this file is: “SDKRoot/Demos SDK/ CaptureAndSend / ca-ES.lproj / Localizable.strings”.

7. CHANGES HISTORY

Date	Changes	Version	Author
May 1th, 2016	- Added Face Detection with Liveness test.	1.4.0.0	Enric
July 7th, 2016	- Added icarLicenseKey property to use all the features of sdk.	1.4.0.1	Enric
August 10, 2016	- Document capture with Autofoto. - Modified DemoSDK with all last functionalities.	1.5.0.0	Enric
August 25, 2016	- Improved Autofoto in user experience.	1.5.1.0	Enric
December 13, 2016	- Added to Localizable.strings all the localizable strings. - Modified back button position to upper left corner in autofoto process.	1.5.1.1	Enric
December 22, 2016	- Sending sdkVersion on IcarServes connections. New custom dialog on autofoto capturing.	1.5.1.2	Enric
December 28, 2016	- Fixed minor bug in AutoFoto process. - Changed the name of the class "NSData+CommonCrypto" to avoid the error of duplicity.	1.5.1.3	Enric
January 24, 2017	- Added new demo sdk. Now there are two samples (AllFeatures and CaptureAndSend). - Change distribution of sdk, before was an static library plus resource directories. Now is a .framework generated with lipo (for simulator and device execution).	1.5.1.4	Enric
February 7th, 2017	- Changed layout of autofoto and face-liveness processes. - Modified some literals strings relationed with autofo and face-liveness.	1.5.1.5	Enric
March 27th, 2017	- For all fields returned by IdCloud_DataResult_Delegate has benn trim spaces from end of string. - Added new function getDeviceInfo on IdCloud_LocalImage. - Added two new fields on IdCloud_Configuration (emailToCheck and phoneNumberToCheck). See section "3.1. CONFIGURATION PARAMETERS" for more information.	1.5.16	Enric
March 30th, 2017	- Fixed memory problems with: CaptureProcess_CAPTURE_FACE_DETECTION (liveness) and CaptureProcess_CAPTURE_IMAGE (autofoto).	1.5.17	Enric
5th of April ,2017	Added new sdk demo called OfflineProcesses.	1.5.18	Enric