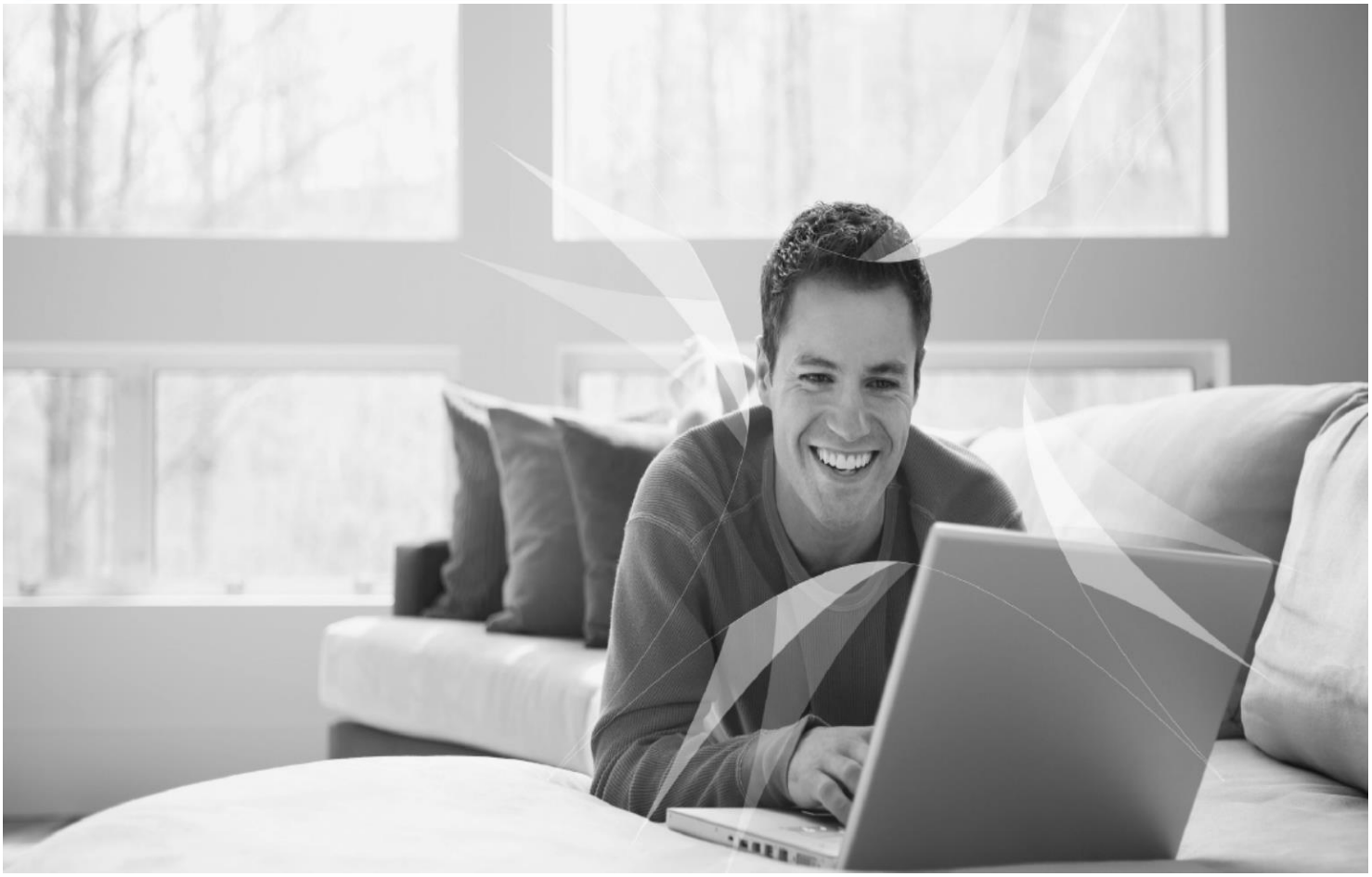
User Guide for Thales Wrapper IDV SDK



FINANCIAL SERVICES & RETAIL

ENTERPRISE

PUBLIC SECTOR

TELECOMMUNICATIONS

TRANSPORT

THALES

Table of contents

Participants

Role	Name
Author	Michail Mavromoustakos

Changelog

Version	Changelog
1.0	Initial Version
1.1	ThalesCamera.js v5.5 ThalesThinLib.js v3.11.3
3.17	Updated with Smart Capture SDK v1.1.6.1
3.18	Updated with Smart Capture SDK v1.1.8.1

1 INTRODUCTION

The Wrapper SDK is a library that abstracts the Web (JavaScript) SDK used for the Thales IDV solution. It provides all interfaces to the underlying SDK, adds logging capabilities and streamlines the integration, however, it doesn't add new functionalities or changes the functionality of the underlying SDK.

Disclaimer: The Wrapper SDK is providing as an integration library free of charge without dedicated support commitments by Thales. The source code can be used as is or part of the source code used or the SDK can be modified and the modified SDK used without any restrictions in terms of software licensing.

2 INSTALLATION

The Wrapper SDK is the "ThalesThinLib.js" provided by your Thales representative.

2.1.1 Supported Browsers and OS

The Wrapper library provides a different API interface, wofklow and logging capabilities without adding any "heavy" logic to the SDKs. As such, it is compabible with all browsers and OS, practically the support of browsers and OS should be taken from the official supported list of Smart Capture SDK or SDKv11.

2.1.2 Loading as Module

ThalesThinLib.js is loaded as a JavaScript module by default.

```
import { ThalesThinLib } from "./ThalesThinLib.js"
```

2.1.3 Loading as a classic Script

The Wrapper SDK can be loaded as a classic JavaScript file by commenting out the last line in ThalesThinLib.js. In that case the global variable window.ThalesThinLib can be used.

```
//*** Comment out this line if you want to load it as a classic JavaScript library
export { ThalesThinLib as ThalesThinLib };
```

2.1.1 Using in Typescript Project

Even if ThalesThinLib.js (Wrapper SDK) is a JavaScript file it can be used in a Typescript project. If loaded as a module:

```
import {ThalesThinLib} from "./scripts/ThalesThinLib.js"
```

if loaded as a classic JavaScript library:

```
(window as any).ThalesThinLib
```

The SDK doesn't export any new types or classes. It uses enumerations that are all under ThalesThinLib.ENUM object and have a type of string. For example:

2.2 SDKv11 INSTALLATION OR MIGRATION

The underlying SDKv11 is found in https://github.com/Acuant/JavascriptWebSDKV11. All the applicable SDK files should be included in the project but the APIs or global functions mentioned shouldn't be used. Specifically:

- Remove any lines of code that load any of the Javascript (.js) files, such as in <script> HTML elements or if loaded programmatically
- Remove any lines in your code containing the objects "AcuantJavascriptWebSdk" or "AcuantCamera" or "AcuantCameraUI" or "AcuantPassivLiveness"
- Remove the global variables "acuantConfig" and "onAcuantSdkLoaded" and global function "loadAcuantSdk()"
- The "acuant-camera" root HTML element could be removed, the Wrapper SDK will automatically create it. Alternatively, it can be left, in which case the Wrapper SDK will use it without modifications
- The "acuant-face-camera-container" root HTML element could be removed, the Wrapper SDK will automatically create it. In this case, it will automatically create also the HTML Element to show the detection hints (face detection results). Alternatively, it can be left, in which case the Wrapper SDK will use it without modifications and will not create an HTML element to show the detection hints

The Wrapper SDK loads all the necessary files, depending on its configuration. For example, if face capture is not required it won't load any face capture related libraries. Note the below:

- Manual Capture, using the native camera, it is not advised to be used for security reasons, so the functionality is not enabled
- The Wrapper SDK has the option to enable "CDN support", but the modification of the URLs via a script if a different hostname than the current page is used is required https://github.com/Acuant/JavascriptWebSDKV11/blob/master/convert_for_cdn.sh

2.3 SMART CAPTURE SDK INSTALLATION OR MIGRATION

The underlying Smart Capture SDK (SDKv12) is shared directly by Thales. It is a package installed with a tool like NPM. The installation guide can be found https://smartcapture-demo.idscan.cloud/demo/sdk-docs/#installation-steps, link is password protected Thales representative can provide the username and password.

Any SDK APIs shouldn't be used. Specifically:

- Remove any lines of code that import the @gbgplc/smartcapture-web module(s). However, install the packages in your project. The Wrapper SDK will import the modules programmatically
- Remove any lines in your code containing the objects "SmartCaptureModule"
- Remove any event listeners on the custom elements "live-face-camera" and "live-document-camera". Even if the HTML elements are left, the event listeners should be removed

- The “live-document-camera” custom HTML element could be removed, the Wrapper SDK will automatically create it. Alternatively, it can be left, in which case the Wrapper SDK will use it without modifications
- The “live-face-camera” custom HTML element could be removed, the Wrapper SDK will automatically create it. Alternatively, it can be left, in which case the Wrapper SDK will use it without modifications

The Wrapper SDK loads all the necessary files, depending on its configuration. For example, if face capture is not required it won’t load any face capture related libraries.

3 INTEGRATION

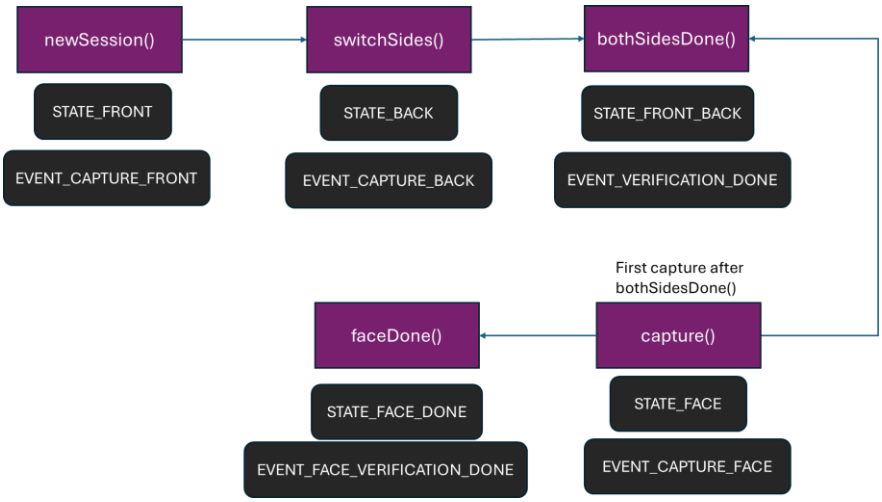
3.1 CAPTURE SESSION

This section assumes that there are 3 images captured. However, it is not mandatory to do so, only document or face capture could be done:

- Front-side image of a document ID is captured
- Back-side image of a document ID is captured (except if it is a passport)
- Face (selfie) image of the person is captured

The session lifecycle is shown below:

- The "state" is an enumeration that can be retrieved at any time from the SDK
- The "event" is the name of "last event" which is recorded in the SDK logs



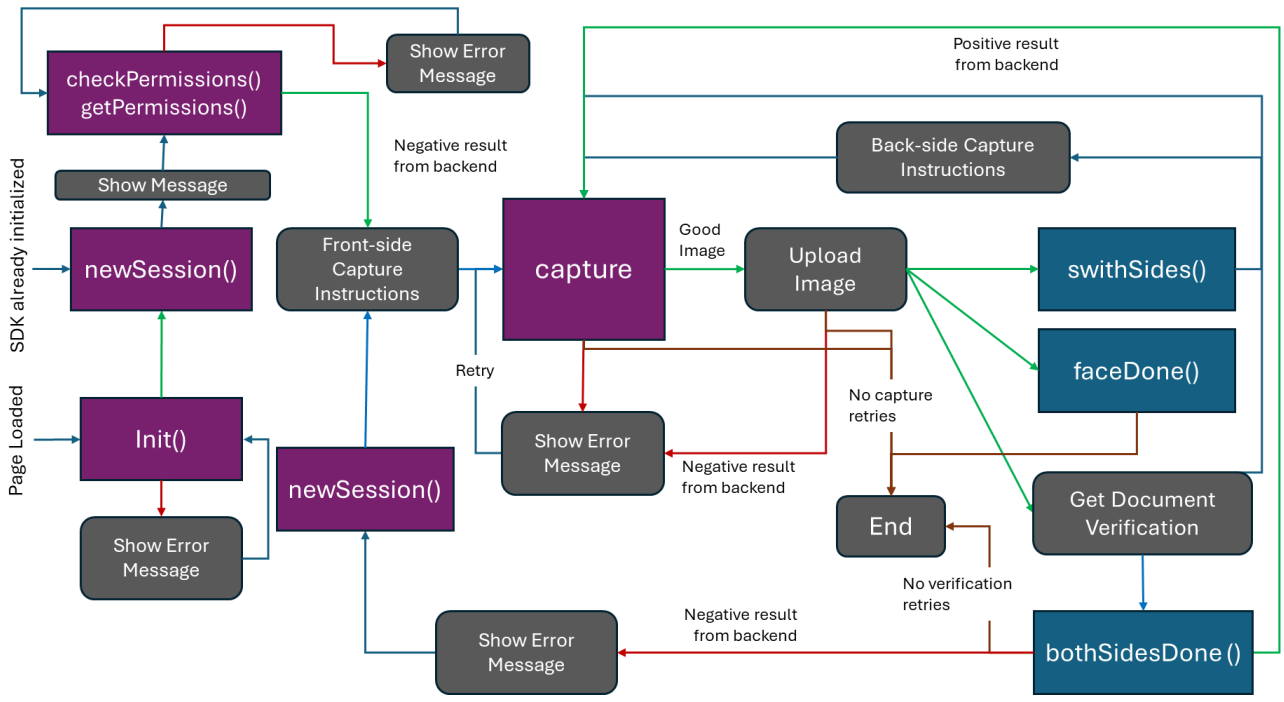
3.2 EXECUTION FLOW

The Wrapper SDK keeps a state of the phase of the identity proofing flow:

1. When the page loads, the SDK libraries are loaded with the init() method. If the SDK is already initialized this can be omitted. Note that this can take a few seconds so it is not recommended to block the UI, rather initialize in the background while the user is reading some instruction messages
2. The user is informed that camera permissions are required. The camera permissions are checked with the checkPermission() method. If not granted, the getPermission() method is called to get the user consent. If not granted, an error message should be shown with instructions on how to grant the permissions. Note that if permissions are not checked, when the SDK is asked to capture an image it will trigger the prompt for a user to accept the permissions. However, if the user denies the SDK will throw an error. As such, it is suggested to check in advance.
3. A new capture session is started with newSession(). This initializes the session object that keeps track of data and results.

4. Instructions should be shown to inform the user to capture the front-side document image. The SDK is used to perform the capture, if successful it will give an image. If not, it will give an error code and an error message can be displayed, prompting the user to retry. After maximum retries, the best image captured in the session will be returned
5. The captured image is uploaded to the backend to perform classification. If the result of classification is valid, then the switchSides() method should be called to tell the SDK that the back-side capture will occur. If the result is negative, a new capture should be performed by going back to step #4
6. Similarly, the back-side is captured and uploaded. After this step is done, the document verification results are asked from the backend and the bothSidesDone() method is called to record the result and indicate to the SDK that document capture is complete. If verification retries are available, a new session should be started with newSession(). The user will continue again from the front-side capture step
7. After document verification is completed, the face (selfie) image can be captured. If the upload is successful or retries depleted, the faceDone() method is called. If not successful, a new capture is performed.
8. During this process, logs can be sent at any time to the backend, the SDK generates a dedicated log object

This is illustrated below, with purple color the SDK methods, with gray color the steps the web app takes:

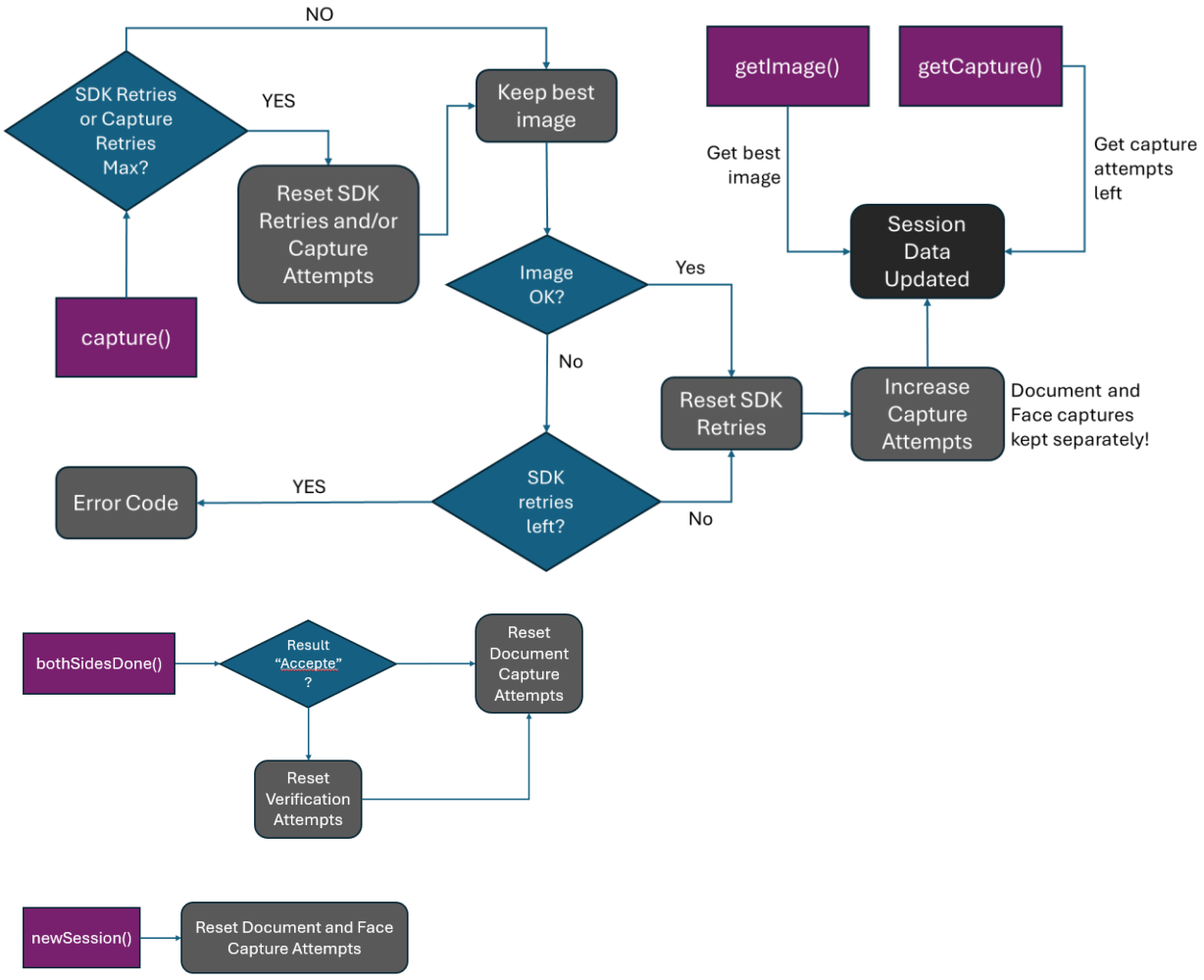


3.1 RETRY HANDLING

The Wrapper SDK handles retry counters on the various actions:

- **Capture Retries:** referred to as "SDK retries". In the event where the image capture is of low image quality or the capture has timed out or the user has pressed the "close" button or due to some unexpected SDK error, the SDK will automatically retry up to the defined "Max SDK Retries" defined. When the retries are depleted, the SDK will give back the best image captured so far
- **Capture Attempts:** when an image is captured (after SDK retries), the image is expected to be uploaded do the backend, since the image can be rejected by the backend, the SDK keeps track of the amount of capture attempts that were uploaded. If the attempts are depleted, the SDK will end the session. The SDK will keep separate retries for document and face image uploads, but will keep a common counter for front-side and back-side captures of the same document
- **Verification Attempts:** when the document images are captured, a verification is performed by the backend. If the verification is rejected, a user can attempt again to capture new front side and back side images. The amount of retries is tracked by the SDK. Note that this is only for document verifications, face verification is done during the upload of the face image

The flow is shown below:



3.2 API DOCUMENTATION

3.2.1 Session Object

The Wrapper SDK stores images captured and logging data into a JavaScript object referred to as a "session". To initialize a session object, use the method below:

```
ThalesThinLib.withSdk.newSession();
```

if the session object needs to be retrieved, i.e. for troubleshooting purposes, it can be retrieved with:

```
const sessionObj = ThalesThinLib.withSdk.getInternalSession();
//If only the size of the object in characters is needed, call instead:
//const size = ThalesThinLib.withSdk.getInternalSession(true);
```

3.2.2 Initialization

NOTE: a few methods of the SDK were accepting a session object as the firt parameter. This will be deprecated so it is not mentioned in teh document, however, the parameter is left

The SDK can be initialized with the below method, by passing a configuration object. Refer to section "Configuration Object" for more details.

```
await ThalesThinLib.withSdk.init(null, config);
```

If the SDK is already initialized, the init() method will just resolve. If there is any error during initializations, the promise is rejected with a ThalesThinLib.ENUM.SDK_INIT_ERROR.

If the configuration itself is invalid, the init() method may return ThalesThinLib.ENUM.SDK_CONFIG_ERROR instead.

It is possible to force a re-initialization by passing the parameter force to 'true':

```
await ThalesThinLib.withSdk.init(null, config, force);
```

Note that SDKv11 (though not Smart Capture SDK) requires a network connection to cross-domain endpoint, which is a Thales' licensing server.

3.2.3 Check Camera Permissions

The SDK requires camera permissions to be granted to control the camera of the device. Two method are provided to check and get permissions, which give a distinct permission status

- Granted: camera permissions are granted
- Rejected: camera permissions were requested but rejected for the given domain or hostname

- Prompt: camera permissions require a user prompt. They are not granted or rejected yet
- Unknown: some error occured while checking or acquiring camera permissions

```
//Check if camera permissions are granted. This will not
//trigger a user message to accept camera permissions but only
//give the current status
const perm = await ThalesThinLib.cameraUtils.checkPermissions()
//returns one of
//ThalesThinLib.ENUM.CAMERA_PERMISSIONS_GRANTED
//ThalesThinLib.ENUM.CAMERA_PERMISSIONS_REJECTED
//ThalesThinLib.ENUM.CAMERA_PERMISSIONS_PROMPT
//ThalesThinLib.ENUM.CAMERA_PERMISSIONS_UNKNOWN


//Ask for permissions. This will trigger the necessary UI for
//a user to accept or reject camera permissions
ThalesThinLib.cameraUtils.getPermissions()
//Alternatively, ask permissions only if not already accepted by
//the user
ThalesThinLib.cameraUtils.getPermissions(true)
//returns one of
//ThalesThinLib.ENUM.CAMERA_PERMISSIONS_GRANTED
//ThalesThinLib.ENUM.CAMERA_PERMISSIONS_REJECTED
//ThalesThinLib.ENUM.CAMERA_PERMISSIONS_PROMPT
//ThalesThinLib.ENUM.CAMERA_PERMISSIONS_UNKNOWN
```

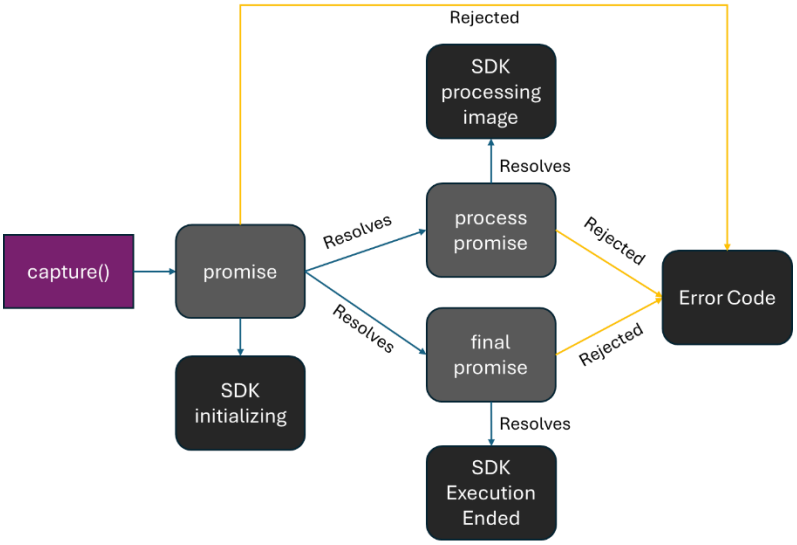
Sample implementation:

```
async function startCameraWithPermissions(isFace, faceAndDoc) {
    let perm = await ThalesThinLib.cameraUtils.checkPermissions();
    if (perm === ThalesThinLib.ENUM.CAMERA_PERMISSIONS_GRANTED) {
        startCamera(isFace, faceAndDoc);
    } else if (
        perm === ThalesThinLib.ENUM.CAMERA_PERMISSIONS_DENIED &&
        ThalesThinLib.platform() !== ThalesThinLib.ENUM.IOS
    ) {
        //Android only supports user enablement form URL icons
        errorMessageUi("Enable camera from URL bar");
        startCameraWithPermissions(isFace, faceAndDoc);
    } else {
        //Inform user that they will be prompted, only if explicitly that is the result
        if (perm === ThalesThinLib.ENUM.CAMERA_PERMISSIONS_PROMPT) {
            errorMessageUi("Please Accept permissions");
        }
        //The user needs to grant permissions or the query permission API wasn't supported
        //and we don't know the status
        perm = await ThalesThinLib.cameraUtils.getPermissions();
        if (perm === ThalesThinLib.ENUM.CAMERA_PERMISSIONS_GRANTED) {
            startCamera(isFace, faceAndDoc);
        } else {
            errorMessageUi("Camera access is required!");
        }
    }
}
```

3.2.4 Start Capture

NOTE: manual capture is going to be deprecated so not mentioned in this document, even if in the code of the current Wrapper SDK and SDKv11

The SDK capature is performed with ThalesThinLib.withSdk.capture() which takes same inputs as the init() method. This result of this method is an object that contains promises to reflect the multiple states of captures. This is shown below:



An example code is shown below

```
        try {
```

```
        //SDK is initializing
        const r = await ThalesThinLib.withSdk.capture(config);
        //The SDK is showing the UI to capture the image
        await r.process;
        //The SDK is processing the image captured
        await r.final;
        //*** If there is an error, the promise is rejected with an exception
        //thrown. Otherwise, there is a valid image to get and send to the
        //backend. Note if retries are depleted, there will be the best
        //image captured in session, but the image maybe null if no image
        //was captured at all (i.e. timeout)
        const image = ThalesThinLib.withSdk.getImage();
        //A specific image can be retrieved by calling
        //getImage(null, SIDE) or getImage(session, SIDE)
        //with SIDE being "FRONT", "BACK" or "FACE"
        //***TBD: Upload image and check classification
        const log = ThalesThinLib.withSdk.getLog();
        //***TBD: Upload log file
      } catch (code) {
        if (code === ThalesThinLib.ENUM.CAMERA_PERMISSIONS_DENIED) {
          //User denied camera permissions. Handle case
        } else if (code === ThalesThinLib.ENUM.ENUM.ERROR_TIMEOUT) {
          //Show dedicated error message (capture timed out)
        } else if (code === ThalesThinLib.ENUM.ENUM.ERROR_CLOSE_BTN) {
          //Show dedicated error message (user pressed "Close Button")
        } else if (code === ThalesThinLib.ENUM.ENUM.ERROR_CROP) {
          //Show dedicated error message
        } else if (code === ThalesThinLib.ENUM.ENUM.LOW_RES) {
          //Show dedicated error message
        } else if (code === ThalesThinLib.ENUM.ENUM.HIGH_GLARE) {
          //Show dedicated error message
        } else if (code === ThalesThinLib.ENUM.ENUM.LOW_SHARPNESS) {
          //Show dedicated error message
        } else {
          //Show generic error message
        }
      }
```

3.2.5 Get Images

The example code is shown in previous section with ThalesThinLib.withSdk.getImage(). Note that the side can be selected as a parameter and the method may return null if no image was captured.

3.2.6 Get Logs

The example code is shown in previous section with ThalesThinLib.withSdk.getLog(). Note tha the result is JSON. Additional data can be used to be included in the SDK logs by calling:

```
ThalesThinLib.withSdk.setLogData({
   key: …,
   value: …,
});
```

with {key, value} being an object with type {string, string}. There are two reserved keys

- ThalesThinLib.ENUM.BACKEND_ERROR, to be used to store an error code from the backend verification server. Note that this **SHOULD NOT contains results such as security tests** but rather a generic error such as “document image upload failed”.
- ThalesThinLib.ENUM.BACKEND_SESSION, to be used to store a session identifier to link the SDK session with a backend session, for example logging the “scenario ID”. Note that this should be omitted if considered not secure doing so.

3.2.1 Get Encrypted Blob

If the Video Injection or Injection Attack Detection solution for face images is used, the input is an encrypted blob instead of an image. That can be retrieved with the below API call instead:

```
//as positive or negative. In this case, it can be passed to the SDK
ThalesThinLib.withSdk.getImage(“FACE”, true)
```

3.2.2 Switch to Back-side

When the front-side is uploaded successfully, the SDK needs to change state. That can be done with the method call below. After this method call, the next capture will store data differently than the front-side data stored:

```
ThalesThinLib.withSdk.switchSides();
```

3.2.3 Check Upload Retries

The SDK keeps track of ”Capture Attempts” to track how many times a captured images is uploaded. Note that if there is no image captured at all, this is still considered a capture.

```
const attObjc = ThalesThinLib.withSdk.getCaptures();
const attemptsLeft = attObjc.left;
const maximumAttempts = attObjc.max;
```

3.2.4 End Document Verification

When the front-side and back-side is uploaded successfully, the SDK needs to change state. That can be done with the method call below. Note that the verification result is passed as either accepted or verified.

The full verification result SHALL NOT be passed back from the backend to teh front-end as that can create a security risk. An attcker can know for example which security tests succeeded/failed.

The SDK needs to know only if the verificaiton was successful and the user should continue on face verification. For example, if chosen to always proceed with face verification, even if document verification was negative, to the SDK the result should be given as accepted:

```
//When the backend verifies the document, it can optionally give back the result
//as positive or negative. In this case, it can be passed to the SDK
const result = verified ? ThalesThinLib.ENUM.VERIFICATION_ACCEPTED :
     ThalesThinLib.ENUM.VERIFICATION_REJECTED;
ThalesThinLib.withSdk.bothSidesDone(null, result);
```

3.2.5 Check Verification Retries

The SDK keeps track of ”Verifications Attempts” to track how many times a document verification occured. This is only useful if there is a flow to allow a user to retry document verification before face verification. If the flow is to end if document verification failed or always proceed to face verification, this can be ommitted:

```
const attObjc = ThalesThinLib.withSdk.getVerifications();
const verificationAttemptsLeft = attObjc.left;
const maximumVerificationsAttempts = attObjc.max;
```

3.2.1 Check State

The SDK keeps track of ”Verifications Attempts” to track how many times a document verification occured. This is only useful if there is a flow to allow a user to retry document verification before face verification. If the flow is to end if document verification failed or always proceed to face verification, this can be ommitted:

```
const attObjc = ThalesThinLib.withSdk.getState();
//returns one of
//ThalesThinLib.ENUM.STATE_FRONT
//ThalesThinLib.ENUM.STATE_BACK
//ThalesThinLib.ENUM.STATE_FRONT_BACK
//ThalesThinLib.ENUM.STATE_FACE
//ThalesThinLib.ENUM.STATE_FACE_DONE
```

3.2.1 Console Logging

The SDK keeps by default uses console logs appending [Thales] in all log entries. This can be disabled by calling:

```
ThalesThinLib.withSdk.setLogger(null, true);
```

The console logs can also be embedded within an HTML element by updating its “value” property, for example using an <input> element. This can be used for troubleshooting purposes when a console log is not available.

3.2.1 Device metadata

The SDK has some utility functions to get some more data from the device. Note that the device model data is included by default in the SDK logs, the list of camera devices not:

```
ThalesThinLib.utils.platform()

await ThalesThinLib.utils.checkDeviceModel()
//Async function returning a string as below with priority
//
// *** Device model string
//The device model if allowed by the browser or OS (not allowed on iOS)
//If not allowed, it will return the below instead:
// *** OS-ScreenWidth-ScreenHeight
//The OS as “ANDROID” or “IOS” or “DESKTOP” and the screen size. This could
//help identify the device model approximately

ThalesThinLib.utils.platform()
//Returns one of the below
//ThalesThinLib.ENUM.ANDROID
//ThalesThinLib.ENUM.IOS
//ThalesThinLib.ENUM.DESKTOP

ThalesThinLib.cameraUtils.getDevices()
//Returns an array of the camera devices names on the attached device
//Should be called after the camera permissions are granted
//for example, after ThalesThinLib.cameraUtils.getCameraPermissions()
```

3.3 Error Codes

All error codes are enumerations (string) starting with ThalesThinLib.ENUM

ThalesThinLib.ENUM.xxxxx	Meaning
ERROR_CROP	Cropping the image failed, the result is that there is no image retrieved. Only applicable for SDKv11. *Note that if an image is captured too far away with manual capture, cropping may succeed but result below the minimum resolution, which is considered a failed crop.*

LOW_RES	The resolution was too low. *Note that for manual tap to capture use case on Smart Capture, this may mean the document was just not present at all, like the corners where not visible.*
LOW_SHARPNESS	Sharpness was below the threshold.
HIGH_GLARE	Glare was detected, below the threshold.
ERROR_TIMEOUT	Glare was detected, below the threshold.
ERROR_CLOSE_BTN	The end user pressed the close button
CAMERA_PERMISSIONS_DENIED	The user has denied camera permissions.
CAMERA_PARAMS_ERR	The camera doesn't support the minimum required parameters for capture (i.e., doesn't support the minimum resolution required).
CAMERA_ISSUE	The SDK failed to start, due to camera related issue.
SDK_FAIL	The SDK failed to start for unkonwn reasons (not due to any of the above)
SDK_ERROR	The SDK failed to start due to some unexpected exception.
BROWSER_NOT_SUPPORTED	The browser is not supported.

In addition, the SDK will keep track of the "worst error", this is the worst error in the sesssion. The priorit of the error codes is shown below:

ThalesThinLib.ENUM.xxxxx	Meaning
IMAGE_UPLOADED	This is the success scenario, an image was uploaded. However, the SDK doesn't know if the image uploaded was successful or not.
LOW_SHARPNESS HIGH_GLARE LOW_RES ERROR_CROP ERROR_TIMEOUT ERROR_CLOSE_BTN BROWSER_NOT_SUPPORTED SDK_ERROR SDK_FAILED CAMERA_PARAMS_ERR CAMERA_ISSUE CAMERA_PERMISSIONS_DENIED CAMERA_PERMISSIONS_UNKNOWN	As expalined in previous table.
NO_CAPTURE	No capture has been performed yet.

SDK_INIT_ERROR SDK_CONFIG_ERROR	As expalined in previous table.

3.4 CONFIGURATION OBJECT

Field	Format	Description
captureTimeout	number	The time in milliseconds the capture will end with ERROR_TIMEOUT. *Default: no timeout, but it is recommended to set 60 seconds*
maxSdkAttempts	number	The maximum image capture attempts for the SDK. *Default: 3*
maxCaptureAttempts	number	The maximum document capture attempts for the SDK. It can be ommited for face image capture only. *Default: 5*
maxFaceCaptureAttempts	number	The maximum face capture attempts for the SDK. It can be ommited for document image capture only. *Default: 5*
maxVerifAttempts	number	The maximum document verification attempts for the SDK. It can be ommited for face image capture only. *Default: 1*
thresholds	object	An object describing the image quality thresholds for document images. It can be ommited for face image capture only.
thesholds.dpi	number or boolean	SDKv11: this is the minimum thresholds of the DPI score *Default: 400* *SDKv12: this indicates if DPI (resolution) should be checked on manual tap to capture* *Default: false*
thesholds.glare	number or boolean	SDKv11: this is the minimum thresholds of the glare score *Default: 50* *SDKv12: this indicates if glare should be checked on manual tap to capture* *Default: false*
thesholds.sharpness	number or boolean	SDKv11: this is the minimum thresholds of the sharpness score *Default: 50* *SDKv12: this indicates if sharpness should be checked on manual tap to capture* *Default: false*

thesholds.minDpi	number [SDKv11]	This is the minimum thresholds of the DPI score, if lower the image is considerd invalied (not cropped) *Default: 250 (ID1 size) / 250 (TD3 size)*
uncropped	boolean [SDKv11]	This gives back the document images uncropped so the server can peform more security tests. *NOTE: On Smart Capture SDK, images are always uncropped, this setting is reduntant. On SDKv11, the document image is cropped both on the SDK side and on server side.*
ui	Object	Sets User Interface related settings

ui.text	Object
	[SDKv11]
		Object with 5 string felds:
		{
		NONE,
		SMALL_DOCUMENT,
		BIG_DOCUMENT,
		CAPTURING,
		TAP_TO_CAPTURE,
		GOOD_DOCUMENT
		}
		The ”NONE” field is displayed when there is no document detected, SMALL_DOCUMENT when document too far away, BIG_DOCUMENT when too close, CAPTURING when document is being captured, TAP_TO_CAPTURE when fallback to tap to captured performed.
		GOOD_DOCUMENT when a good quality document is detected, leave ’null’ to show a countdown.
		Default: ”ALIGH” for NONE, ”MOVE CLOSER” for SMALL_DOCUMENT, ”TOO CLOSE” for BIG_DOCUMENT, ”CAPTURING” for CAPTURING and a 3 second countdown for GOOD_DOCUMENT

ui.generalInfoTexts	Object
	[Smart Capture]
		Object with 4 string fields:
		{
		front, back, fallbackFront, fallbackBack
		}
		where ”front” is the text shown to guide the user when capturing the front side document, ”back” the text when capturing the back side document, ”fallbackFront” the text guiding the user when the manual tap to capture is enabled for the front-side capture and similarly ”fallbackBack” for the back side manual tap to capture.
		Default:
		”Position your Identity Document within the fram” for generalInfoTexts.front and generalInfoTexts.back and ”Position your Identity Document within the frame and select the capture button.” for generalInfoTexts.fallbackFront and generalInfoTexts.fallbackBack

ui.autoCaptureText	string [Smart Capture]	The text of the auto capture toggle button and reflected on the very top text (on top of the video stream), with ”alertText” is the the text below that. *Default: ”Auto Capture”*
ui.alertText	string [Smart Capture]	The text below the autoCaptureText on the very top, hinting to the user to use auto capture if wanted. *Default:”Struggling to capture? Disable Auto Capture”*

ui.moveCloserHint	Object
	[Smart Capture]
		Object with 2 string fields:
		{
		title, description
		}
		The ”title” shows message next to the ”alert exclamation icon” close to the video stream, where the ”description” is the longer text showed outside the video stream preview.
		Default: ”Document is too far” for the title and ”Fit the document fully within the frame.” for the description.

ui.fixBlurHint	Object [Smart Capture]	Like above, but for the text related to blurriness issues. *Default: ”Document is blurry” for the title and ”Hold the document and device steady.” for the description.*
ui.fixGlareHint	Object [Smart Capture]	Like above, but for the text related to glare issues. *Default: ”Glare Detected” for the title and ”Move the document away from direct light sources.” for the description.*
ui.outOfFrameHint	Object [Smart Capture]	Like above, but for the text related to the document not being in the frame or being too far away. *Default: ”Document not in frame” for the title and ”Fit the document fully within the frame.” for the description.*
ui.capturingHint	Object [Smart Capture]	Like above, but for the text displayed to guide the user when there is no issue mentioned above. *Default: ”Document detected” for the title and ”Capturing” for the description.*
ui.successTime	number [Smart Capture]	The number of milliseconds to add an artificial delay after the image is captured, to enhance potentially UX. *Default: 500*
ui.showHelpIcon	boolean [Smart Capture]	If set to ’false’, it will hide the help icon ”?”. Note that as of version 1.1.6.x the help menu is not localized, if non-Enlgish language is required this should be set to ’true’ to hide the menu. *Default: true*
faceNotFoundHint	[Smart Capture]	Message shown during face detection if face is not found. *Default: ”Face not found”*
probabilityTooSmallHint	string	Message shown during face detection if face is probably not found.

		Default: "Probabilty too small"
tooManyFacesHint	[Smart Capture]	Message shown during face detection if more than one face is detected. *Default: "Too many faces"*
faceAngleTooLargeHint	string	Message shown during face detection if face has an angle (user is not looking straight). *Default: "Face not straight on"*
faceTooSmallHint	[Smart Capture]	Message shown during face detection if face too far away from the device. *Default: "Face too far"*
faceCloseToBorderHint	string	Message shown during face detection if face is too close to the device. *Default: "Face too close"*
autoDetectConfig	Object	Define settings on the auto capture.
autoDetectConfig. **disableToggle**	boolean [Smart Capture]	If set to 'true', the auto capture toggle will never appear. *Default: false (auto capture toggle enabled)*
autoDetectConfig. **showToggleTimeout**	number [Smart Capture]	Timeout in milliseconds, when elapsed the auto capture toggle is shown. Before that, it is hidden. If not set, auto capture toggle is shown right away. *Default: null (auto capture is shown right away)*
autoDetectConfig. **enableToggleTimeout**	number [Smart Capture]	Timeout in milliseconds, when elapsed the auto capture toggle is enabled. Before that, it is "grayed out". *Default: 15,000 (15 seconds)*
autoDetectConfig. **maxExpTime**	number [Smart Capture]	Maximum time for a detection event to occur to consider the detection too slow. This is meant to balance behavior on low-end devices that can be too slow to perform automatic capture by quantifing what "too slow" means. *Default: 3,000 (3 seconds)*
autoDetectConfig. **maxAllowedTooLong**	number [Smart Capture]	If set, this indicates how many "too slow detections", as determined from "maxExpTime", will force the Smart Capture SDK to enable auto capture toggle button. Note that if "disableToggle" is set to 'true', then this can be ignored. *Default: null (disabled)*
fillRootColor	string [SDKv11]	The background color, any Javascript definition of color would work for example "rgb(255,255,255)" or "black".

actionButton	Object
	[SDKv11]
		An object to add a close button overlay:
		{
		text,
		top, bottom, left, rigth,
		textSize, textColor, textShadow
		}
		whereas the "text" field is the text shown (by default "CLOSE"), the "top, left, right, bottom" is the position of the HTMLElement (uses fixed position),

		the "textSize" and "textColor" is the size and color of the text and "textShadow" the text shadow applied. All are optional. *Note: Smart Capture SDK has by default such a button*
init	object	Series of settings that are not meant to change after the SDK initializes.
init.onlyDocument		SDK will capture only a document image *NOTE: If false or not present and onlyFace is false or not present, then both document and face images are captured.*
init.onlyFace	boolean	SDK will capture only a face image. *NOTE: If false or not present and onlyDocument is false or not present, then both document and face images are captured.*
init. **smartDocumentCapture**	boolean	If 'true', use Smart Capture SDK. If 'false', use SDKv11 *NOTE: if ommitted, SDKv11 is used*
init.debugBorders	boolean	For debugging purposes, draws a border around the HTML elements drawn. *Default: false*
init.slowLoadTimeout	number [Smart only]	The milliseconds the initialization method should wait until the libraries are initialized. This doesn't include loading files (downloading the files), but only their actual initialization. If not set, the SDK will not wait for initialization to be completed before ThalesThinlib.withSdk.init() resolves, but rather when ThalesThinLib.withSdk.capture() is called there can be some additional waiting time before first capture. If the value is set an initialization exceed the timeout, the ThalesThinLib.withSdk.init() will be rejected with error ThalesThinLib.ENUM.SDK_INIT_ERROR. *Default: null (no timeout)*
init.isDebug	boolean [Smart Capture]	Sets the Smart Capture SDK in debug mode. This enables the "development mode" of the face capture component of the SDK. Note that if "onlyFace" is set to 'true', this will be ignored. *Default: false*
init.baseSdkPath	string [SDKv11]	The absolute or relative path to load the SDK files *Default:* current page URL (href) appended with "/webSdk/dist"
init.jpegQuality	number	From 0 to 1.0, a decimal number indicating the JPEG quality (compression rate). *Default: 0.9, note that this is different from the default of the SDKv11 which is 1.0*
init.basicAuth **init.jwt**	String [SDKv11]	SDKv11 authentication (for licensing purposes) string. If basicAuth is set, this should be the Base64(username:password) string. If jwt is set, this shoud be the JWT retrieved from the backend.
init.endPoint	String [SDKv11]	The endpoint to initalize the SDKv11. This should be the full endpoint starting with "https://....".

init.maxRetries	3 [SDKv11]	The number of times the SDKv11 will try to initialize (authenticate) with the backend licensing server (ACAS). If it fails to initialize after maxRetries the ThalesThinLib.withSdk.init() method resolves into an exception of ThalesThinLib.ENUM.SDK_INIT_ERROR *Default: 3*
init.retryTimeout	500 [SDKv11]	The time in milliseconds to wait between retries to initialize the SDKv11. *Default: 500*
init.fromCDN	Boolean [SDKv11]	If set to true it will support loading the SDK from a CDN, this refers to loading the SDK from a different domain than the current page is. *Default: false*
init.slowMode	Boolean [SDKv11]	If set to true the SDKv11 will execute the processing of the image slower, but use less memory. This is referred to as using a ”single worker”, as otherwise by default it woudl use two workers, one to crop the image and one to analyze the glare and sharpness. *Default: false*
Init.channel	string	Information on the ”business channel” or ”business use case”, to be used for logging purposes only.

3.5 SMART CAPTURE FACE ANTI-DEBUGGING PROTECTION

This section applies ONLY:

- Smart Capture SDK
- Only if the face capture module is used
- Only if ”Video Injection” or ”Injeciton attack detection” services are used

3.5.1 Introduction

Browsers come with powerful built-in developer tools that allow to manipulate elements on the web page, including network communication, overloading standard browser APIs, inspectings line by line the web page’s source code and so on. To avoid a more sophisticated attack where an attacker is tampering with the web page execution, with the intent to inject the seflie image, the SDK includes anti-debugging protection features.

This includes several layers of protection, such as code obfuscation, execution flow timing, infinite loops to block debugging tools and so on. Given the development tools are used for development, this features needs to be taken into account during the software development cycle.

3.5.1 Production vs Development mode

The SDK by default runs in ”production mode” with anti-debugging protection enabled. The SDK can run in development mode with anti-debugging protection disabled by setting the face component attribute ”environment” to the value of ”development”. The attribute can be set to ”production”, even if this is the default behavior.

If the face component is set via HTML:

```
<!-- Development Mode -->
<live-face-camera environment="development"></live-face-camera>

<!-- Production Mode -->
<live-face-camera environment="production"></live-face-camera>
```

If the face component is programmatically created, the environment attribute (note it is an attribute and not a property) can be set as:

```
const liveFaceCamera = document.createElement("live-face-camera");
if (isDebug) {
     liveFaceCamera.setAttribute('environment', 'development');
 } else {
     liveFaceCamera.setAttribute('environment', 'production');
 }
```

3.5.2 Production deployment Considerations

To further incrase security, the development mode should be entirely disabled in production. The SDK allows this by having a seperate development and production library. The development library SHALL NOT be included into a production environment, as that would allow more easily an attacker to manipulate the web page code to enable development mode on the SDK. This can be achieved in two ways:

- **Single Package:** the zip file containing the SDK will include the development and production library. If deployed in proudction, the develoment can be removed from production via scripting. The development library path is the whole directory with a relative path (i.e. starting from node_modules):

\@gbgplc\smartcapture-web\node_modules\idlive-face-capture-web-development

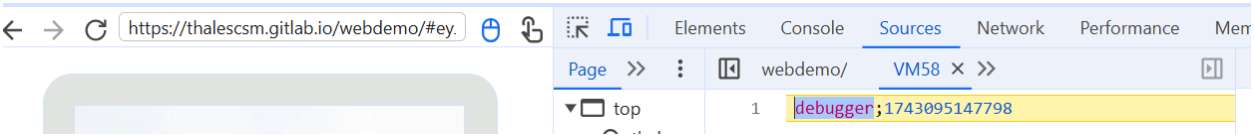
- **Two packages:** a seperate zip file containing the SDK with the development library removed can be installed in production, whereas the zip including the development library installed in test environemnts. A sample such zip with the development library removed is included with the appendix _NoDevMode. It is noted that the only difference is the development library removed.

3.5.3 Development Considerations

When the SDK is loaded in production mode it will enable anti-debugging protection for the whole page it runs. Before the SDK is loaded, this mechanism is not enabled. This means if the SDK is loaded on a page and other components need to be tested, the anti-debugging mechanism may interfere with them. The development mode can be used to avoid this.

It is noted that if development tools are not enabled, the SDK will run as normal. For example, if console logs were saved into a file, that file could be reviewed, but if the development tool of the browser were enabled to view the console, the anti-debugging proteciton feature would interfere.

For reference, if anti-debugging mode is enabled and the browser development tools are used, the behavior would look like below:



Note that even if the code execution was to continue or even if the line showed upon was commented out by using browser overrides, the SDK will still try to block the debugger, by trying to go to an infinite loop and make the browser tab eventually unresponsive or detect that there is a slowness and not show the camera and so on. For security evaluation purposes the anti-debugging protection can be tested, but for development purposes it should not be attempted to be bypasses as it can lead to an undefined behavior.

3.5.4 Security Evaluation Considerations

To evaluate the overall security, the Production mode should be used. This will allow to evaluate what an attacker would have to bypass. However, it is noted that the development mode still performs all security tests to provide injection attack detection which is evaluated by the backend. This means the development mode can be used for security evaluations, as long as it is understood that it is subject to attacks based on development tools.

4 SDK Logs and Reporting

The SDK creates session logs that allow to gather all necessary data in order to:

- **Conversion rate analysis** (Funnel Analysis): know where the user has abandoned during the process and why
- **Root cause analysis:** easier identify client-side issues by capturing error and device information in a uniform way
- **SDK logs:** enhance client-side logging with specific SDK logs

4.1 Session Log Details

Note that the fields with "Back" and "Face" in the end refer to data for the back-side and selfie/face image respectively

Field	Format	Description
id	Object	The unique identifer of the transaction
id.id	string	The unique ID of the user/device, the SDK attempts to makes this persistent across retries by storing it in the local storage.
id.sessionId	string	An ID of the session. It will change if newSession() is called or if face verification completes after calling faceDone() or if there is only document verification and bothSidesDone() is called.
id.channel	number	The channel information in the configuration object.
lastEvent	string	Tha name of the last event, refer to section "Capture Session" for the available event names
timestamp	number	The timestamp (epoch time) of the last event reflected in the logs.
device	Object	Device information
device.platform **device.model**	string	The platform returned with ThalesThinLib.withSdk.platform() method call and model returned with ThalesThinLib.withSdk.checkDeviceModel() method call
device.userAgent	string	The user agent retrieved with navigator.userAgent
device.brands	array [Object]	An array of objects as: { brand, version} where "brand" is the brand name of the browser and "version" the version of the browser. This is retrieved with navigator.userAgentData.brands
device.attrs	string, number or object	Additional device attributes. It could be expanded in a future version.
externalData	array [Object]	An array of objects as {key, value}. Stored with ThalesThinLib.withSdk.setLogData()

version	string	Format of ”X.Y.Z”, the version of the Wrapper SDK. Used for reference only.
internalSdkVersion **internalSdkVersionFace**	string	The Smart Capture or SDKv11 version. This is recorded if available on the underlying SDK. It is recorded both for document and face components seperatly.

options	Object	{
		thresholds, captureTimeout, slowMode, smartCapture, jpegQuality
		}
		Record important configuration options, with all values having the same name as the configuration object.

sdkAttempts	number	The remaining SDK attempts
maxSdkAttempts	number	The maximum SDK attempts
captureAttempts **captureFaceAttempts** **verificationAttempts**	number	The number of images captured (excluding SDK retries) for documents (captureAttempts) and faces (captureFaceAttempts) as well as the document verification (verificationAttempts) performed.
maxCaptureAttempts **maxFaceCaptureAttempts** **maxVerificationAttempts**	number	The maximum configured value for the above
verificationResult **verificationFaceResult**	string	The document and face verification result, as recorded with bothSidesDone() and faceDone() method calls.
isUncropped **isUncroppedBack**	boolean	If the front and back image was uncropped. Only useful for SDKv11
worstError **worstErrorBack** **worstErrorFace**	string	The worst error captured so far for the front-side, back-side and face image respsectively. Refer to ”Error Codes” section.
lastError	string	The last SDK error or null if the last capture has no error.

captureDetails
captureDetailsBack
captureDetailsFace
	Object	Object for the details on the captured front-side, back-side and face images:
		{
		width, height, sharpness, glare, dpi, isPortraitOrientation, size, extra
		}
		where ”width” and ”height” is the width and height of the image, ”sharpeness” is the sharpness score of SDKv11 or ”true” if the Smart Capture SDK detected a sharp image, similarly ”glare” and ”glare”, ”isPortraitOrientation” is a boolean indicating if the image was captured while the device in portrait mode, ”size” the total size of the image

captureDetails.extra.captureMode captureDetailsBack.extra captureMode captureDetailsFace.extra captureMode	string	If automatic capture then the value is ThalesThinLib.ENUM.AUTO_CAPTURE if tap to capture then ThalesThinLib.ENUM.TAP_TO_CAPTURE.
captureDetails.extra.captureTime captureDetailsBack.extra captureTime captureDetailsFace.extra captureTime	string	The total time in milliseconds it took to capture all images in the session. This is to reflect how much time in total the end user spent capturing the front-side, back-side and face images.
captureDetails.extra.processTime captureDetailsBack.extra processTimeBack captureDetailsFace.extra processTimeFace	string	The maximum time in milliseconds for the SDK to process the image after capture, this is applicable only for SDKv11.
captureDetails.extra.croppingRatio captureDetailsBack.extra croppingRatio captureDetailsFace.extra croppingRatio	number [SDKv11]	Decimal number indicating the ratio the image was cropped, from the one captured
captureDetails.extra.isPassportSize captureDetailsBack.extra isPassportSize captureDetailsFace.extra isPassportSize	boolean	'true' if it is a passport size

errors	Object	An object with a format of:
		{
		errorName1: N,
		errorName2: N,
		errorName3: N,
		}
		Where "errorNameX" is the name of the error and "N" the number of times teh error appear. There is no explicit definition of errors, this can contain more error information like a browser exception name.

autoDetection autoDetectionBack	Object	An object of: {

		count, start, end, maxTime, tooLong, failledChecks, state, lostFocus } when ”count” the number of detection events, ”start” and ”end” the starting and ending time, ”maxTime” the maximum time before detection events, ”tooLong” how many times it exceeded a defined timeframe between detection events, ”failedChecks” is the detection errors.
camera	Object	
isDetectInit	boolean	The detection engine for face capture was initialized. Only useful for Smart Capture and only for face capture.
initTime	number	The number of milliseconds it took to initialize the SDK, excluding the face detection initialization.
uiShown	boolean	’true’ if the SDK UI was shown to the end user.

5 WEB FRAMEWORK GUIDELINES

This section gives an overview of tested JavaScript frameworks. **It is NOT a list of compatible or "supported" or "certified" frameworks.** Any framework can work with the SDK, this explains some particularities in terms of integration from our experience gathered from multiple projects.

Framework	Remarks
Angular	No known issues/limitations.
React	No known issues/limitations.
Standard WebView	No known issues/limitations except. - Android: Some versions of WebView cannot apply zoom. The SDK appies zoom to accomodate some knoew devices issues on autofocusing (a few iOS, Galaxy S and N series). This limitation may apply to other frameworks relying on WebView
Ionic	No additional known issues/limitations
Ionic Native	Older Ionic Native frameworks may have issues on loading "tiny_face_detector_model-shard1" file. To solve for that, the file should be renamed as "tiny_face_detector_model-shard1.bin". Then inside the "tiny_face_detector_model-weights_manifest.json" the line "paths":["tiny_face_detector_model-shard1"] should be changed to "paths":["tiny_face_detector_model-shard1.bin"] (adding .bin)
Flutter	No known issues/limitations using flutter_inappwebview 5.x or later except: - Permissions have to be set by using the onPermissionRequest function. Permissions should be automaitcally granted if the app has permissions to use the camera
React Native	No known issues/limitations.

5.1 CAMERA PERMISSIONS FOR ANDROID NATIVE

Native apps require to provide camera permissions on the app level. Each OS will specify which permissions are required for "WebRTC" or "Media Stream" or "Camera". By default, the WebView will check for user permissions in addition to the app permissions. It is recommended to silently grant without asking the user again, after checking that the permissions have actually been granted. Otherwise, silently denying permissions.

If the WebView grants permissions but they are not granted on the app level, the result should be considered undefined.

```
mWebView.setWebChromeClient(new WebChromeClient() {
    @Override
    public void onPermissionRequest(final PermissionRequest request) {
        if (hasPermissions(mContext, PERMISSIONS)) {
            request.grant(request.getResources());
        } else {
            request.deny();
        }
    }
```

```
private static boolean hasPermissions(Context context, String[] permissions) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M && context != null && permissions !=
null) {
        for (String permission : permissions) {
            if (ActivityCompat.checkSelfPermission(context, permission) !=
PackageManager.PERMISSION_GRANTED) {
                return false;
            }
        }
    }
    return true;
}
```