HOWTO

V8 - 2024-04-12

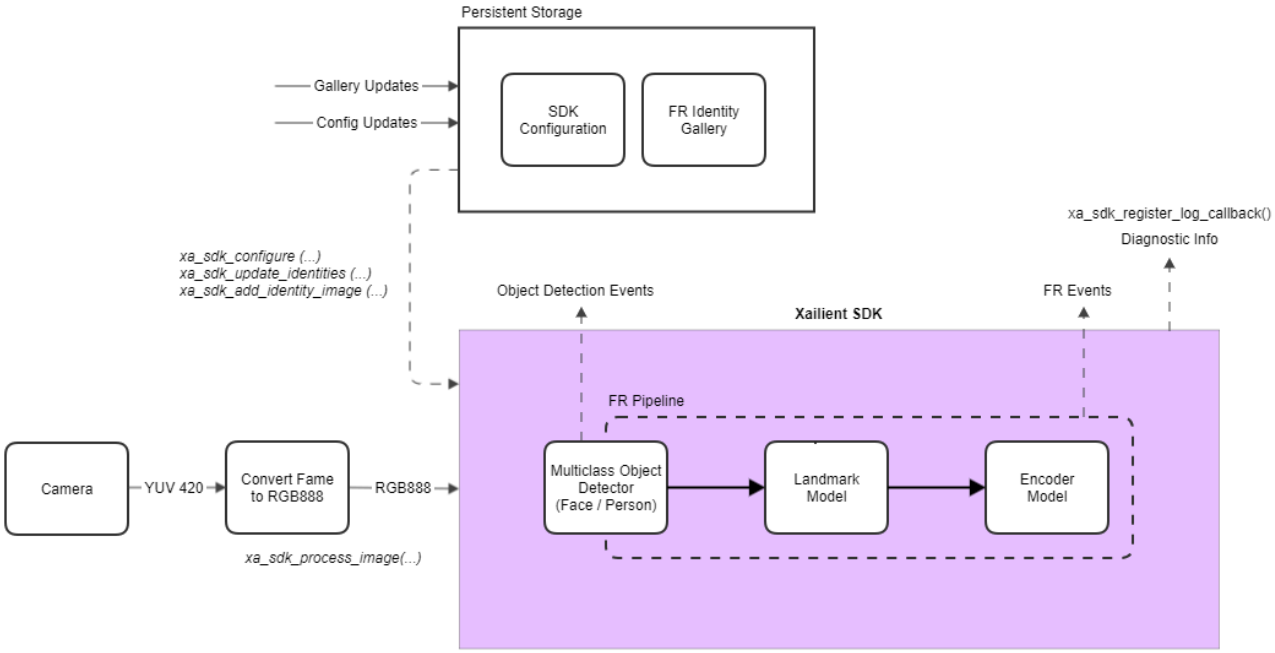
This document outlines how to integrate against the Xailient Face Identification SDK and libraries.

Nomenclature

- Brand Customer - the organization desiring to integrate Xailient software into their product.
- Device Checkin - periodically sending Device Checkin Data from the Device to the cloud.
- Device Checkin Data - data for the Device Checkin, including SDK ID, SDK Version, Vision Cell ID, Vision Cell Version, Vision Cell Identities, Errors, Warnings.
- Device Integrator - those responsible for providing backend services to the Device. This is generally not Xailient, and is often not the device manufacturer; this is often is the Brand Customer or partner.
- Face Embedding - processed data derived from an image necessary for Face Identification software. These are stored in Gallery Identities.
- Face Track Event - events around a face
  - Detected
  - Recognized
  - Not recognized
  - No longer detected
- FI SDK (Face Identification Software Development Kit) - the binaries and headers that provide Face Identification functionality
- Gallery Identities - json specifying all identities and Face Embeddings a device should use for face identification. The device generates Face Embeddings for registration from the Gallery Identity Manifest and often persists this on the device.

- Gallery Identity Manifest - json specifying all identities and images a device should use for face identification. Orchestrait generates this for device consumption. The SDK augments this with Face Embeddings and other data.
- Identity - the information associated with a “person” that the facial recognition software will identify. This is made of an identifier, a version, a name, and Face Embeddings
- Vision Cell - part of an SDK responsible for running inference on an image and generating events

SDK Context



SDK Integration

The Xailient Face Identification interface is comprised of two .so libraries and some header files.

They require g++ version 5.4.0

They require libstdc++ version 6.0.21

A Note on Receiving Values from Pointer to Pointer Arguments

If an SDK argument is a pointer to pointer - e.g. `const xa_sdk_json_blobs_t * * json_blobs` - the idiom is to create a `const xa_sdk_json_blobs_t * my_json_blobs` and pass the address of it to the function - e.g. `&my_json_blobs`. After the function call, inspect the pointed to values.

Important! Pointers in the returned structs are only valid until the next call to a function in the SDK.

A Note on JSON

SDK functions receive JSON sent from Orchestrait. While persisting this JSON, be careful to not modify it in any way. Converting between JSON and strings can mistakenly insert escape characters in front of double quotes depending on the specific method used. Modifying this JSON in such a way will lead to errors.

SDK Directory structure

For SDKs distributed as a shared library, the libraries and executables must be organized in a specific file structure for the binaries and libraries to find each other:

```
/
/lib
/lib/libxailient-fi.so
/lib/libxailient-fi-vcell.so
/bin
/bin/<application>
```

Note this doesn't apply to SDKs distributed as an archive file (*.a) and statically linked.

Gallery Identity Update Methodology

1. `xa_sdk_update_identities`
   a. Pass in the Gallery Identity Manifest. This "returns" identityID/imageID pairs of images the SDK needs to generate Face Embeddings and updated Gallery Identities suitable for passing to `xa_sdk_update_identities`
2. The device acquires an image detailed in one of the identityID/imageID pairs. When the device has an image available:
3. `xa_sdk_add_identity_image`
   a. Pass in the image information. This "returns" identityID/imageID pairs of images the SDK needs to generate Face Embeddings and updated Gallery Identities suitable for passing to `xa_sdk_update_identities`
4. Continue registering images until the remaining identityID/imageID list is empty.
5. If registering the new identities returns no error, persist the new Gallery Identities JSON object.

Important! Pointers in the returned structs are only valid until the next call to a function in the SDK. Don't loop over an old copy of the identityID/imageID pairs after calling xa_sdk_add_identity_image().

Order of Calls to SDK

1. `xa_sdk_initialize`
   a. For shared library SDKs, pass in the path to vision cell if it's different than the default
2. `xa_sdk_configure`
   a. Pass in the locally stored configuration options as JSON, if any
3. `xa_sdk_update_identities`
   a. Pass in the JSON object containing identities
4. `xa_sdk_process_image`
   a. Pass in the image (encoded as RGB888)
   b. Parse the response to send results to the cloud

Things to persist / read-write out to disk

Configuration - JSON sent from Orchestrait to the customer cloud by `POST /deviceConfigurationChanged` detailed below. This JSON gets passed through "as is" to `xa_sdk_configure()`

Gallery Identities - JSON created on the device from a Gallery Identity Manifest sent from Orchestrait to the customer cloud by `POST /deviceConfigurationChanged` detailed below. This JSON gets passed through "as is" to `xa_sdk_update_identities`.

Securing Face Embeddings

Since Face Embeddings, which are encapsulated in the Gallery Identities json mentioned above, are considered PII they must be secured on the device.

To be compliant with most privacy regulations the Face Embeddings must be encrypted when stored persistently on the device. It is the responsibility of the Brand Customer to ensure that this is occurring since the Xailient FR SDK does not have access to any PKI infrastructure for accessing encryption keys, rotating keys, handling key revocations, etc.

While the Xailient FR SDK is not responsible for persisting Face Embeddings there are some best practices that the Brand Customer should consider.

- Encryption must be done using a well known secure algorithm such as AES (using at least 128 bit keys)
- If the device hardware has tamper resistant hardware (e.g. TPM) that supports securely storing encryption keys then that should be used to store encryption keys.
- If the Brand Customer has access to PKI infrastructure from the device then that should be utilized to secure the PII on the device.
- If no such support exists on the device and the OS on the device supports a file system with permissions then the encryption key(s) should be put in a file/directory that requires elevated permissions to access.

Steady State things to do

1. Periodically (approximately once per day) or when the configuration or gallery changes, do a Device Checkin with json from `xa_sdk_get_device_checkin_json()`
2. Receive a configuration update from the cloud if available, persist it, and pass to `xa_sdk_configure()`
3. Receive a Gallery Identity Manifest from the cloud if available, persist it and follow the Gallery Identity Update Methodology
4. Invoke Face Identification through `xa_sdk_process_image()` when a new image is available and parse results from `xa_sdk_json_blobs_t` to pass information to the cloud as applicable
   a. Face Track Events
   b. Accuracy Monitoring

Other things to do

1. Register a callback to `xa_sdk_register_log_callback()` and report log messages up to the cloud

Thread Safety

SDK calls are NOT thread safe - do not call SDK functions concurrently.

Code Examples

xa_sdk_initialize

```
xa_fi_error_t returnValue;

const char * path_to_vision_cell = < initialize to the path of the vision cell > // For shared lib
returnValue = xa_sdk_initialize(path_to_vision_cell);                            // For shared lib

returnValue = xa_sdk_initialize();                                               // For static lib

if (returnValue != XA_ERR_NONE) {
    < handle errors >
}
```

xa_sdk_configure

```
const char * configuration = < initialize to the persisted configuration json as a null-terminated c string >
```

```
returnValue = xa_sdk_configure(configuration);
if (returnValue != XA_ERR_NONE) {
    < handle errors >
    return;
}
```

xa_sdk_update_identities

```
const char * gallery_identities = < initialize to the persisted gallery identities json as a null-terminated c s
const xa_sdk_identity_images_t * remaining_identity_image_pairs;
const char * updated_json_identities;
returnValue = xa_sdk_update_identities(gallery_identities, &remaining_identity_image_pairs, &updated_json_identi
if (returnValue == XA_ERROR_NONE) {
    < see xa_sdk_update_identity_image example for an approach to updating the gallery  - this initialization wo
}
else
{
    < handle errors >
    return;
}
```

xa_sdk_update_identity_image

```
const char * galleryIdentityManifest;
const xa_sdk_identity_images_t * remaining_identity_image_pairs;
const char * updated_json_identities;
xa_fi_error_t returnValue;

if (< a new gallery manifest exists >) {
    galleryIdentityManifest = < retrieve the Gallery Identity Manifest from the cloud >;

    // Step 1
    returnValue = xa_sdk_update_identities(galleryIdentityManifest,
                                           &remaining_identity_image_pairs,
                                           &updated_json_identities);
    if (returnValue == XA_ERR_NONE) {
        < persist updated_json_identities >
    }
    else {
        < handle errors >
    }

    // Step 2 / Step 4
    while ((returnValue == XA_ERR_NONE) && (remaining_identity_image_pairs->number_of_remaining_images > 0)) {
        < acquire the image for remaining_identity_image_pairs->identity_images[0] >
        xa_fi_image_t image = < convert the acquired image to xa_fi_image_t - see fi_image.h >

        // Step 3
        returnValue = xa_sdk_add_identity_image(remaining_identity_image_pairs->identity_images[0].identity_id,
                                                remaining_identity_image_pairs->identity_images[0].image_id,
                                                &image,
                                                &remaining_identity_image_pairs,
                                                &updated_json_identities);
        // Step 5 - persist after each image to avoid needing to download them again
```

```
        //          this step could also be placed after the while loop
        if (returnValue == XA_ERR_NONE) {
            < persist updated_json_identities >
        }
        else {
            < handle errors >
        }
    }

    const char * deviceCheckinJson = xa_sdk_get_device_checkin_json();
    < perform a deviceCheckin with the deviceCheckinJson >
}
```

xa_sdk_process_image

```
xa_fi_error_t returnValue;
xa_sdk_process_image_outputs * process_image_outputs;

if (< a new image is available >) {
    returnValue = xa_sdk_process_image(&image, &process_image_outputs);

    if (returnValue == XA_ERR_NONE) {
        for (int index = 0; index < process_image_outputs->number_of_json_blobs; ++index) {
            xa_sdk_json_blob_t * blob = process_image_outputs->blobs[index];

            if (blob->blob_descriptor == XA_FACE_TRACK_EVENT) {
                < send blob->json to Face Track Event endpoint >
            }
            else if (blob->blob_descriptor == XA_ACCURACY_MONITOR) {
                < send blob->json to Accuracy Monitor endpoint >
            }
            else {
                < handle errors >
            }
        }
    }
    else {
        < handle errors >
    }
}
```

xa_sdk_get_device_checkin_json

```
if (< it's time once a day >) {
    const char * deviceCheckinJson = xa_sdk_get_device_checkin_json();
    < perform a deviceCheckin with the deviceCheckinJson >
}
```