DeepAffex™ Developers Guide

This developers's guide will help you effectively use NuraLogix™ DeepAffex™ technologies. Here you will find step-by-step instructions to get you started and also more detailed references as you delve deeper.

The Introduction illustrates the principle behind Transdermal Optical Imaging and explains the DeepAffex™ architecture.

Chapter 2 walks through a very simple example that can extract facial blood-flow, send it to the DeepAffex™ Cloud for processing and display results. Chapters 3 and 4 go into more details about the DeepAffex™ Cloud API and Extraction SDK respectively.

Chapter 5 showcases the capabilities of the DeepAffex™ Dashboard.

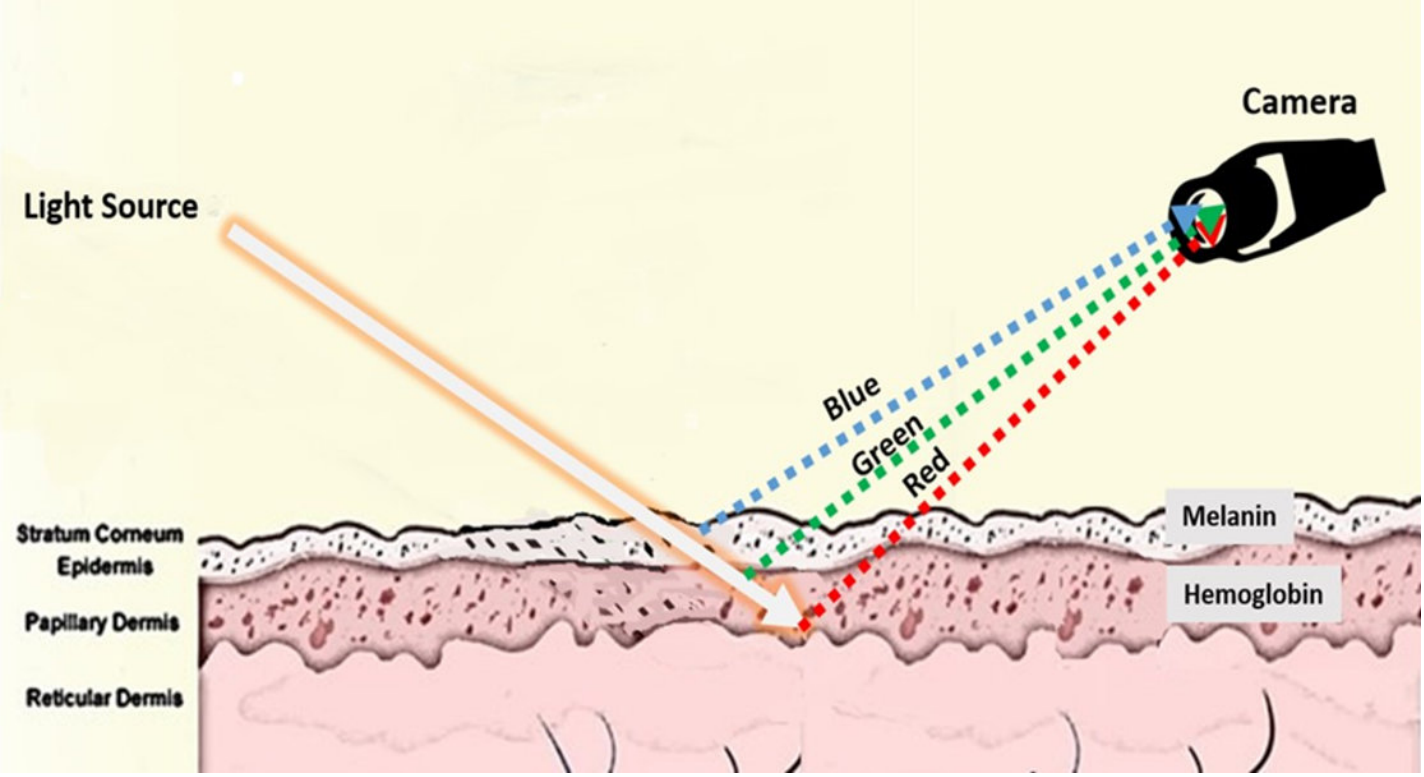
Last updated on 2021-12-17 by Rajas Sambhare (v1.3.1)

Introduction

Transdermal Optical Imaging (TOI™)

Transdermal Optical Imaging (TOI™) is a technology developed by NuraLogix™ that can be used to extract blood flow information from a sequence of images (typically from a video stream of a person's face.)

Recall that a digital image consists of several color channels. As a person's heart beats, the color of their skin undergoes minute variations. TOI™ captures and amplifies these variations and uses them to extract blood flow information. When this is done from biologically significant regions of interest of the face, the result is called facial blood-flow information.



There are three pre-requisites to extracting useful facial blood-flow information:

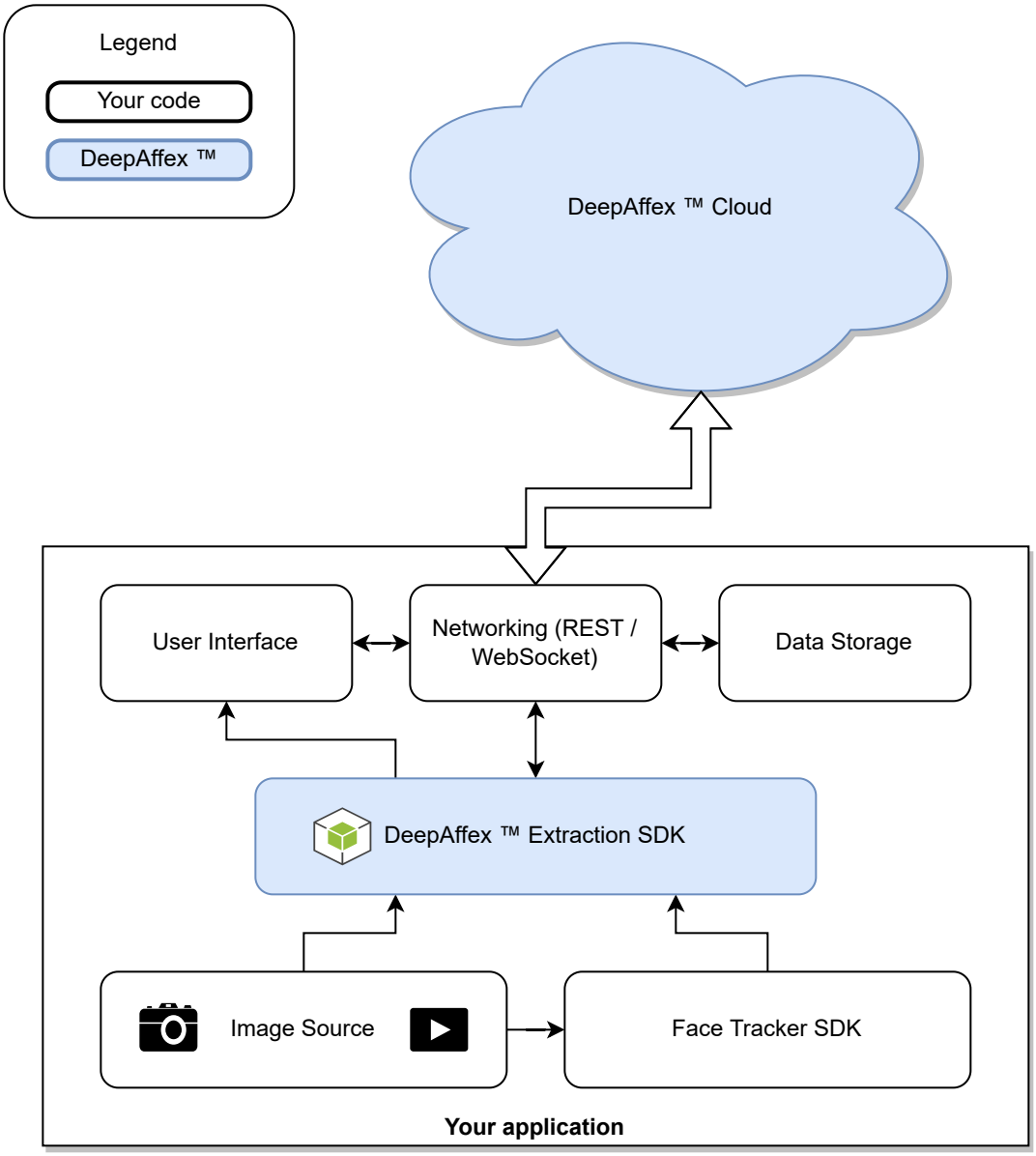
1. A continuous *sequence of images* of a person's face (typically produced by a camera or a video file)
2. Accurate *image timestamps* (typically produced by the camera or read from the video file) and
3. Facial feature *landmarks* (typically produced by a face tracking library)

DeepAffex™

DeepAffex™ is NuraLogix's overarching solution for extracting and processing facial blood-flow information.

You can extract facial blood-flow information using the DeepAffex™ Extraction SDK. Then you can send it to the DeepAffex™ Cloud to estimate several biological signals such as heart rate, heart rate variability and more. NuraLogix™ has also developed several neural network models that can also predict biosignals like blood pressure or health markers like stress index.

The overall architecture of a DeepAffex™-based application is shown below:



In addition to the DeepAffex™ Extraction SDK, you will need a method of producing a sequence of images (a camera or a video file,) a face-tracking engine and networking code that communicates wih the DeepAffex™ Cloud, either using HTTP REST or WebSockets.

The next chapter walks through a simple Python example that can extract facial blood-flow, send it to the DeepAffex™ Cloud for processing and display results.

Getting started

This chapter walks through `dfxdemo` - a simple Python-based demo that demonstrates how to use the DeepAffex™ SDK and DeepAffex™ Cloud API.

`dfxdemo` can extract facial blood-flow from a video file or webcam, send it to the DeepAffex™ Cloud for processing and display the results. It can also be used to display historical results. In this chapter, we will focus on video files.

To begin, please clone the demo from its GitHub repo and follow the instructions in the README.

Understanding `dfxdemo`

`dfxdemo` has top-level commands that roughly correspond to the way the DeepAffex™ Cloud API is organized. It uses dfx-apiv2-client-py to communicate with the Cloud.

Device tokens and user tokens

Authentication on the DeepAffex™ Cloud API uses JSON Web Tokens. Each token contains key information pertaining to the current access request. All tokens are mapped against an internal policy manager that specifies what access levels it has. The token policies control every access to the API. There are two types of tokens:

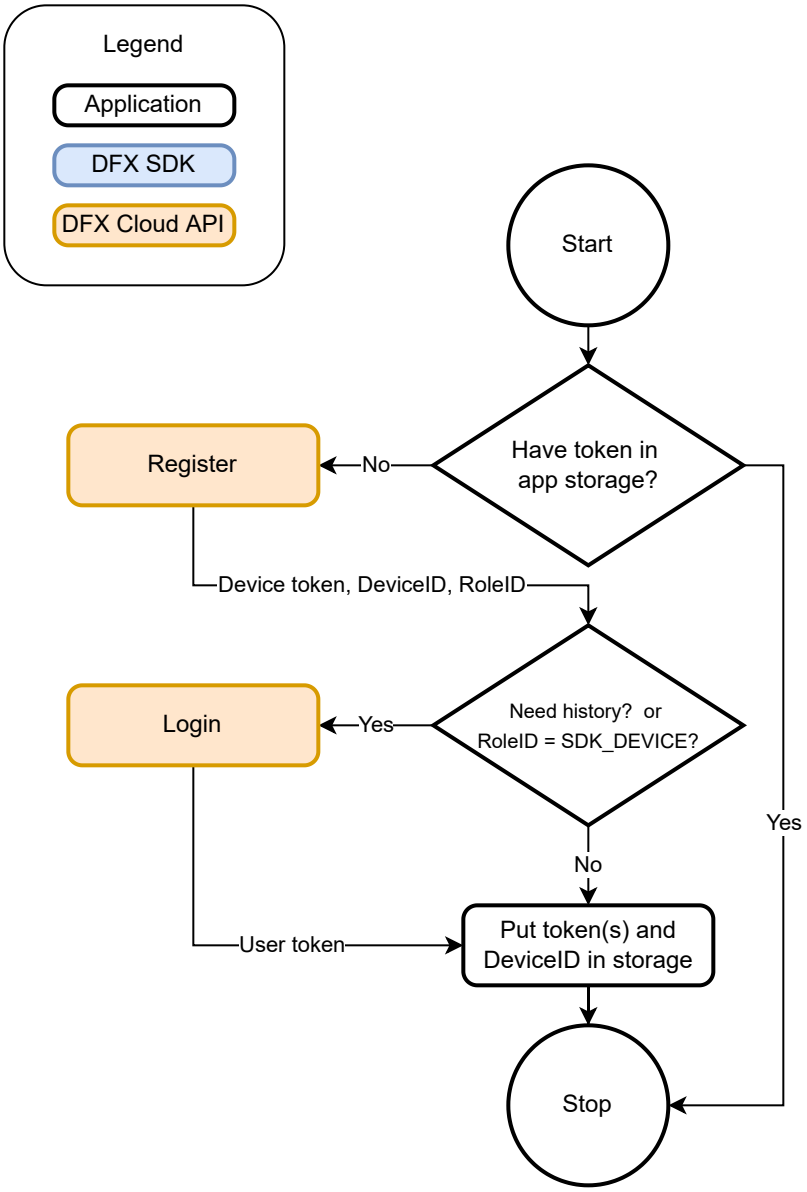
A *device token* usually represents a single device. A device token is permanent and provides access to endpoints that allow for device-specific features e.g. making measurements. To obtain a *device token* for the device in use, register your organization license on the DeepAffex™ Cloud.

```
python dfxdemo.py org register <your_license_key>
```

A *user token* provides access to most non-measurement related endpoints. To obtain a *user token*, login as a user:

```
python dfxdemo.py user login <email> <password>
```

Note: Obtaining a *user token* is mandatory, if the response to register returns a `RoleID` of `SDK_DEVICE`.



By default, `dfxdemo` stores tokens in a file called config.json. In a production application, you will need to manage all tokens securely.

Note: All the commands below, use the tokens obtained above as illustrated on line 71.

DFX Points and Studies

The DeepAffex™ Cloud organizes around the concept of Points and Studies.

A *DFX Point* is a biosignal of interest that is assigned a unique name on the DeepAffex™ Cloud. (e.g. HR_BPM is the DFX Point for heart rate.)

A *DFX Study* is a collection of DFX Points that are computed in one measurement. A Study is based on a *Study Template* which determines which Points it contains. You can list the Studies

available to you, retrieve a Study's details and select one for use in measurements.

```
python dfxdemo.py studies list
python dfxdemo.py study get <study_id>
python dfxdemo.py study select <study_id>
```

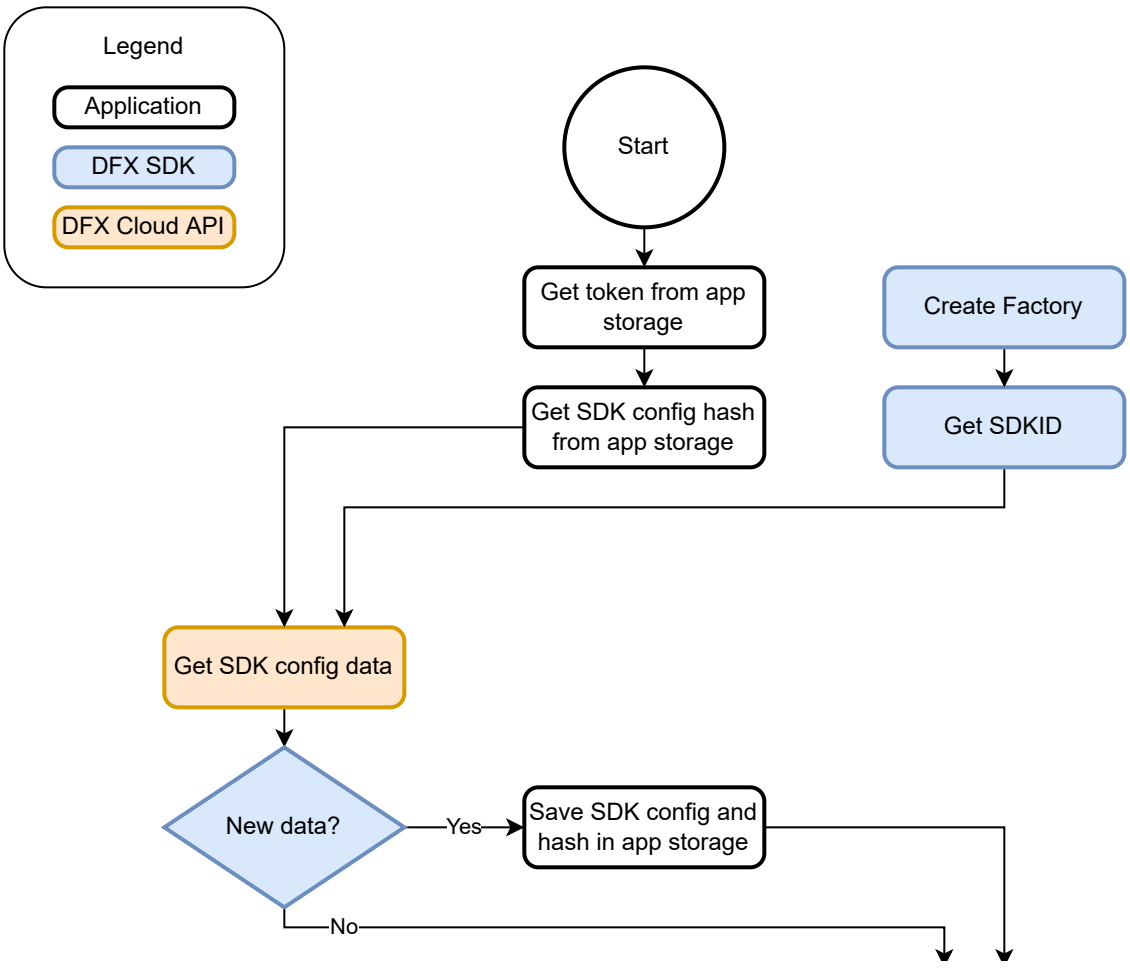
Making a measurement

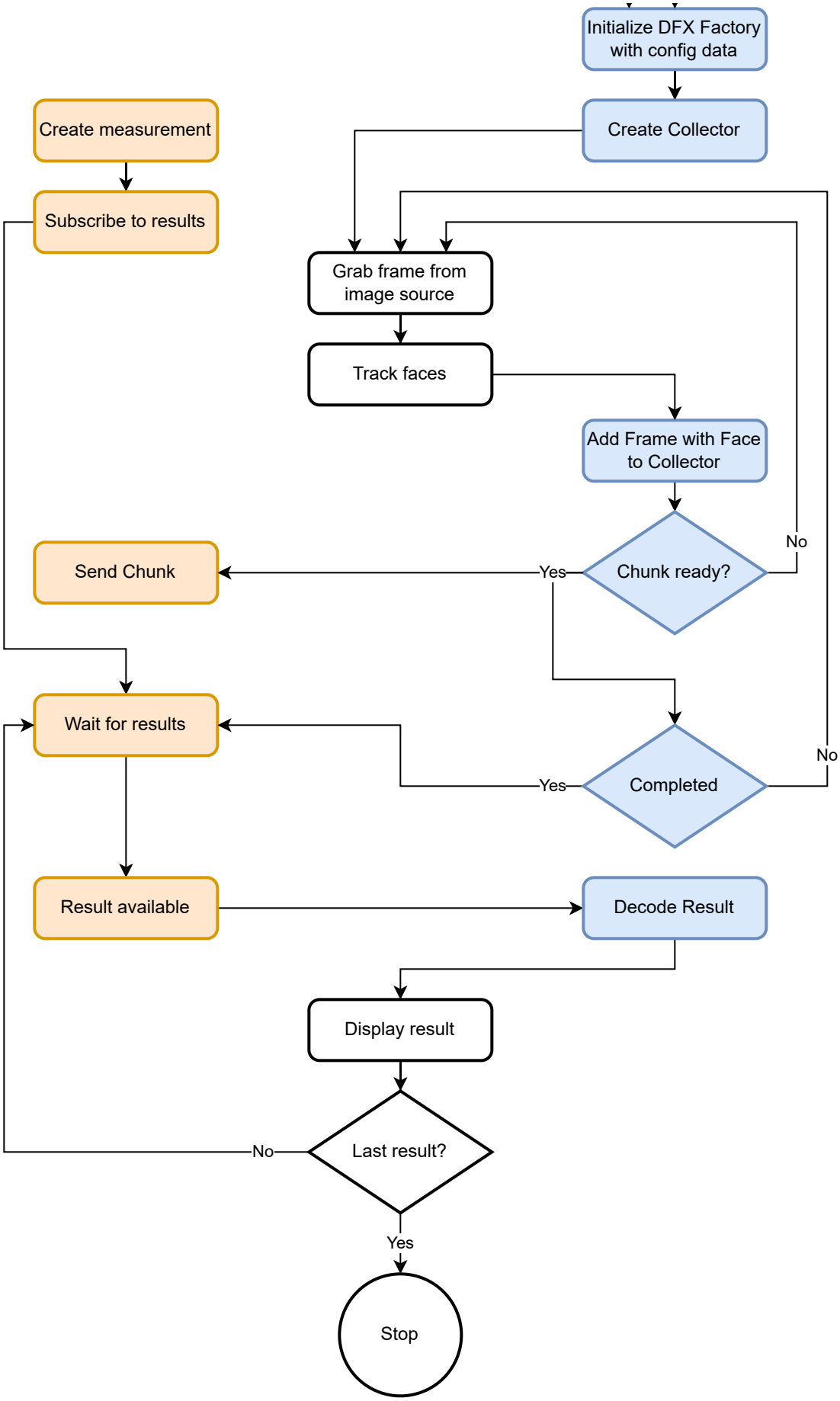
The process of extracting facial blood-flow from a sequence of images and sending it to the DeepAffex™ Cloud for processing is called *making a measurement*.

`dfxdemo` uses OpenCV to read individual frames from a video, Dlib to track facial landmark features in each frame and libdfxpython (DFX SDK's Python bindings) to extract facial blood-flow. To make a measurement using the selected study:

```
python dfxdemo.py measure make /path/to/video_file
```

The facial blood-flow data from a video is sent to the DeepAffex™ Cloud in fixed duration chunks (5 seconds by default) over a WebSocket. As the measurement progresses, accumulated results are sent back over the same WebSocket and displayed. When the last chunk is received by the DeepAffex™ Cloud, the overall results are computed and returned.




Initialize DFX Factory with config data
Create Collector
Create measurement
Subscribe to results
Grab frame from image source
Track faces
Add Frame with Face to Collector
Chunk ready?
No
Yes
Send Chunk
Wait for results
Completed
No
Yes
Result available
Decode Result
Display result
Last result?
No
Yes
Stop

Measurement results and history

Historical measurement results associated with a user and their details can also be retrieved using `dfxdemo`:

```
python dfxdemo.py measurements list
python dfxdemo.py measure get <measurement_id>
```

These results are also available on the DeepAffex™ Dashboard which includes a sophisticated graphical display.

Next steps

In the next chapter, we will discuss the DeepAffex™ Cloud API in more detail.

DeepAffex™ Cloud

The DeepAffex™ Cloud is used to analyze the facial blood-flow extracted by the DeepAffex™ Extraction SDK. It can also be used to manage users and studies under your organization.

The DeepAffex™ Cloud API is organized into endpoint groups composed of individual endpoints. e.g. `General`, `Users`, `Profiles`, `Measurements` etc. are high-level groups of calls while `General.Status` is an individual endpoint within the `General` group.

The API is canonically documented on the DeepAffex™ API Apiary.

Transports

The DeepAffex™ Cloud API endpoints are available using HTTP and WebSocket transports. The DeepAffex™ Cloud does not limit access across different transports, so you may use the transports interchangeably.

API endpoints are transacted using RESTful patterns with the HTTP transport using JSON encoded data and the Websocket version using Protocol buffers to encode data. Going forward, we will refer to "HTTP-based access using RESTful patterns" simply as the REST API and the WebSocket equivalent as the WebSocket API.

The REST API is available at: https://api.deepaffex.ai while the WebSocket API is available at: wss://api.deepaffex.ai. (For compatibility reasons, you can also use ports 9443 and 9080 for REST and WebSocket respectively but for new applications please use the default port - 443.)

Please note that by default you will be automatically routed the server nearest to your geographical location. If you wish to connect to a specific region, please contact NuraLogix™ for the correct URL for your geography.

An important difference between the REST and WebSockets API is that the `Measurements.Results` ("subscribe to results") endpoint is only available on the WebSocket API. This endpoint is used to get real-time intermediate and final results of an active measurement over a bidirectional WebSocket connection.

WebSockets

This transport method is highly optimized and provides a fast and efficient method of communicating with DeepAffex™ Cloud. We recommend using the WebSocket API for platforms where real-time results are required.

In order to maintain the connection with the Cloud, you must implement proper ping/pong heartbeats as required by the WebSocket standard specification. In addition to the standard use of WebSocket connectivity mechanics, the DeepAffex™ Cloud uses a custom request/response exchange pattern outlined below.

Request structure

A WebSocket request has to be an buffer, with the following structure:

```
Buffer( [ string:4 ][ string:10 ][ string/buffer:variablelength ] )
```

The first 4 bytes are an *Action ID* for the request, which is defined in the Endpoint Summary Table.

The next 10 bytes are a *Request ID* - a string identifier unique for each request. The client is responsible for creating a unique Request ID. It could be a counter or a randomly generated alphanumeric string e.g. "JDUEHDJEKO" or "000000001". This ID is only used within the current connection and does not interfere with other concurrent connections using the same token. Request IDs are not stored, so reusing them is possible however not recommended.

The remaining bytes are the body of the request and has to be a serialized Protobuf. The DeepAffex™ Protobuf definitions are available for download on the DeepAffex™ website.

Response structure

Requests that do not abide by the structure defined above are not processed and a response with the following body is returned:

```
{
  Code: "INCORRECT_REQUEST"
}
```

Appropriately structured requests, get responses with the folowing structure:

```
Buffer( [ string:10 ][ string:3 ][ string/buffer:variablelength ] )
```

The first 10 bytes are the *Request ID* matching a request to a response.

The next 3 bytes are an HTTP status identifier. Status codes are defined in the Header Response Codes Table.

The remaining bytes are the response body as a serialized Protobuf. You can deserialize it using the DeepAffex™ Protobuf definitions mentioned above.

The client needs to manage requests in memory and await the response asynchronously. The DeepAffex™ Cloud will always respond to a request to close it out, regardless of if there is an error or not. In addition, to the status code, the body will also contain relevant error information.

Some details

An important difference in the call methods is translating query string and parameters that would typically be sent via the REST service to the WebSocket API. Outlined below are

properties when making a request to the `Meta.retrieveByType` endpoint:

```
{
  "Foo": "bar",
  "Query": {
    "Namespace": "all",
    "Fields": "Foo,Bar"
  },
  "Params": {
    "Type": "Group",
    "ID": "7ab8bee7-808c-4123-9811-e385adb8c77d"
  }
}
```

In essence `Query` and `Params` are values that are extracted from the payload body and interpreted as values that would typically be expected by a REST request.

Error messages are transported in turn in similar object formats as listed below. The WebSocket interface also follow this pattern.

Note: Many endpoints use arrays in the root of the payload. Unfortunately Protobufs do not support rooted arrays, hence the value needs to be wrapped in an object. The DeepAffex™ Cloud looks for the `Values` property in a payload and handles the containing array as a root array. The following is an example of a request to `Groups.addUsers`:

REST request example:

```
  [
      "7ab8bee7-808c-4123-9811-ef85adb8c77f",
      "7ab8bee7-808c-4123-9811-ef85adb8c75f",
      "7ab8bee7-808c-4123-9811-ef85adb8c71f",
      "7ab8bee7-808c-4123-9811-ef85adb8c72f",
      "7ab8bee7-808c-4123-9811-ef85adb8c77f"
  ]
```

Protobuf wrapped example of the same payload:

```
{
  "Values": [
      "7ab8bee7-808c-4123-9811-ef85adb8c77f",
      "7ab8bee7-808c-4123-9811-ef85adb8c75f",
      "7ab8bee7-808c-4123-9811-ef85adb8c71f",
      "7ab8bee7-808c-4123-9811-ef85adb8c72f",
      "7ab8bee7-808c-4123-9811-ef85adb8c77f"
  ]
}
```

First steps

Upon connecting to the DeepAffex™ Cloud, we recommend checking the current API version and general status. This is done by calling the `General.Status` endpoint which will respond with the appropriate status details. This endpoint does not require any authentication.

For most other endpoints though, an appropriate Authorization header must be supplied. This is discussed in the next section.

Authentication

Appropriate authentication is necessary to call most of the endpoints available on the DeepAffex™ Cloud.

Tokens

Authentication on the DeepAffex™ Cloud uses JSON Web Tokens. Each token contains key information pertaining to the current access request. All tokens are mapped against an internal policy manager that specifies what access levels it has. The token policies control every access to the API. There are two types of tokens:

- A *device token* usually represents a single device. A device token provides access to endpoints that allow for device-specific features e.g. making measurements. To obtain a *device token* for the device in use, register your organizations License Key on the DeepAffex™ Cloud using the `Organizations.RegisterLicense` endpoint.
- A *user token* provides access to most non-measurement related endpoints. To obtain a *user token*, login as a user using the `Users.Login` endpoint.

Once you have obtained a token, it needs to be supplied in the header of all subsequent API calls as shown below:

```
Authorization: Bearer [token]
```

For WebSockets, if the supplied authorization details are incorrect, the connection will be refused and the socket dropped by the server. Upon a successful connection the socket will open and handshakes should be complete, signalling availability of the service.

More details and best practices

- Since tokens represent authentication information, please make sure to **store them securely** - typically in some form of secure storage.
- Some License Keys have a limited number of allowed active Devices. If you need to use your License on a different device, make sure to unregister your current device first by calling the `Organizations.UnregisterLicense` endpoint.

- The response to `Organizations.RegisterLicense` contains a `RoleID` for the license - if it is `SDK_DEVICE`, then obtaining a *user token* is mandatory.

- Device tokens may expire. To ensure that a token is valid, you can call the `General.VerifyToken` endpoint. If the `ActiveLicense` field in the response is false or if the API returns one of the following errors: `INVALID_TOKEN`, `LICENSE_EXPIRED`, `INACTIVE_LICENSE` or `INACTIVE_DEVICE`, then the client must re-register the license.

Organizations

In the DeepAffex™ Cloud's business-to-business services model, the Organization is the partitioning "silo" that gets created and licensed by a business.

All measurements, results, raw data etc. that are created under an Organization are secure and isolated from other Organizations. Only NuraLogix™ can create Organizations.

These series of endpoints are all related to managing Organization data. We will discuss two of them in more detail here. The rest are detailed on the DeepAffex™ API Apiary.

Registering Devices

As mentioned before, a *device token* has to be obtained before a measurement can be made from a device. To obtain a *device token* for the device in use, the `Organizations.Register` endpoint has to be called.

This endpoint is called with the License `Key` for your organization. Other parameters in the request are `DeviceTypeID`, `Name`, `Identifier` and `Version`. On a successful request, the DeepAffex™ Cloud responds with a *device token*, `DeviceID`, `RoleID` and `UserID`.

Please ensure that both the `DeviceID` and *device token* are **securely stored** on the device as both are needed for un-registering a device.

If the `RoleID` is `SDK_DEVICE`, then this *device token* has to be used to obtain a *user token* using the `Users.Login` endpoint before measurements can be made.

The token (device or user) then has to used in *all* subsequent calls to the DeepAffex™ Cloud.

Un-registering Devices

In some cases, there are a limited number of devices allowed on a License Key. You can however, un-register a device and gain one License use back. To un-register a device, you can call the `Organizations.Unregister` endpoint with the *device token* and the `DeviceID`.

Note that this does not delete any historical results that were made with this device.

Re-registering Devices

Device tokens may expire. To ensure that a token is valid, you can call the `General.VerifyToken` endpoint. If the `ActiveLicense` field in the response is false or if the API returns one of the following errors: `INVALID_TOKEN`, `LICENSE_EXPIRED`, `INACTIVE_LICENSE` or `INACTIVE_DEVICE`, then the client must re-register the license using the `Organizations.Register` endpoint described above.

Users, Profiles and Anonymous Measurements

Users

Endpoints in this group are related to managing users associated with an Organization.

User Login

Users have to login and obtain a *user token* before they can make measurements that will be associated with them. This useful for getting historical results which can be used to track long term health.

You can use the `Users.Login` endpoint to login using an email address and the `Users.RequestPhoneLoginCode` and `Users.LoginWithPhoneCode` to login using a phone number.

User Logout

There is no explicit endpoint for user logout. However simply destroying the *user token* locally is enough to be considered a logout.

Other endpoints in this group are discussed in more detail on the DeepAffex™ API Apiary.

Profiles

By default every measurement is accompanied by user and device identifiers, which were used to create it. However, there are scenarios where this is not enough.

One such scenario would be a user using the same device to make a measurement on behalf of someone else. For instance, an individual making a measurement for a family member or a researcher making measurements on multiple subjects.

User Subprofiles using `UserProfileID`

In order to address such scenarios and further differentiate measurements within the same user/device pair, a `UserProfileID` can be used. Profiles are linked to a user.

There's no upper limit on a number of profiles for given user, and DeepAffex™ Cloud API can be used to manage a full lifecycle of those using the `Profiles` endpoints

Anonymous Measurements

Though the DeepAffex™ Cloud API provides a complete set of methods to maintain end user accounts, some API consumers ("partners") might prefer to not use these facilities for reasons like privacy concerns or already having a database of users and not wanting to maintain an external copy of it. In such situations you can create anonymous measurements without obtaining a *user token*, using just a *device token* instead.

Anonymous Measurements using `PartnerID`

However, without a link between the partner's users and their anonymous measurements, any analysis of such measurements will not be possible. This link can be created by using a `PartnerID` - a 2-48 character long string, that can be added to the `Measurements.Create` endpoint.

`PartnerID` can hold a unique-per-user identifier, or any other value which could help to reconcile measurements with users later using the `Measurements.List` and `Measurements.Retrieve` endpoints.

For some billing plans (e.g. pay-per-anonymous-user) providing a `PartnerID` may be mandatory.

Studies

As mentioned before, the DeepAffex™ Cloud organizes around the concept of Points and Studies.

A *DFX Point* is a biosignal of interest that is assigned a unique name on the DeepAffex™ Cloud. A *DFX Study* is a collection of DFX Points that are computed in one measurement. A Study is based on a *Study Template* which determines which Points it contains. A *Study Template* (and thus any studies derived from it) has a *Study Type* which is a broad categorization of templates.

For example, a *Study Type* of `HEALTH` could have associated with it two *Study Templates* - `HEART_HEALTH` and `STRESS`. A study called `ELDERLY_HEART_HEALTH` could then be created from the `HEART_HEALTH` template.

The endpoints in this group are related to Study management.

CRUD

Studies can be created, retrieved, updated and deleted using the `Studies.Create`, `Studies.Retrieve`, `Studies.Update` and `Studies.Delete` endpoints respectively.

Study Configuration Data

A *Study* has associated with *configuration data* which is needed to initialize a `Factory` in the DeepAffex™ Extraction SDK. This data can be retrieved using the `Studies.Sdkconfig` endpoint.

To call this endpoint, you have to pass the `SDK ID` of the SDK, the `Study ID` and the hash of the current data on hand. (The hash can be an empty string if calling for the very first time.)

If there is updated configuration data available, the Cloud will respond with HTTP status 200 and the body will contain the Base64 encoded configuration data and its hash. If hash sent was up to date, then the Cloud will respond with HTTP status 304.

Please cache the study configuration data and its hash for future use with the SDK for this Study.

Measurements

The process of extracting facial blood-flow from a sequence of images and sending it to the DeepAffex™ Cloud for processing is called *making a measurement*. The results of this process is called *Measurement*.

Historical results

A history of the measurements made by a device, user or profile is available on the DeepAffex™ Cloud. You can retrieve this history by using the `Measurements.List` endpoint.

The list can be filtered by `Date`, `EndDate`, `UserProfileID`, `UserProfileName`, `PartnerID`, `StudyID`, `StatusID` or any combination thereof.

A measurement can have one of the following statuses:

Status	Description
CAPTURING	A new record was created and results are being received
PROCESSING	Capture is complete and the record is processing
COMPLETE	The analysis is complete and ready for consumption
ERROR	An error occurred during processing
INCOMPLETE	Capturing process returned as incomplete/not enough data

Detailed results of a single measurement can be retrieved by using the `Measurements.Retrieve` endpoint.

Making a measurement

Making a measurement involves three (or two) different endpoints - `Measurement.Create`, `Measurement.AddData` and, if real-time results are desired, `Measurements.Subscribe`.

Creating a measurement

`Measurement.Create` is the first endpoint that you must call - it will prepare the DeepAffex™ Cloud for receiving data and return a Measurement ID.

Subscribing to results

In order to receive real-time results, you will need to touch the `Measurements.Subscribe` endpoint. This is WebSocket only endpoint. You must pass a unique *Request ID* to it - the results will be sent back with the same *Request ID* on the WebSocket as they become available on the Cloud.

Adding data

Once you have subscribed, you can send data to the Cloud using the `Measurements.AddData` endpoint. The bytearray containing the facial blood-flow data that is sent to the Cloud (a "chunk"), is produced by the DeepAffex™ Extraction SDK. If you are using the REST API, then this array needs to be Base64 encoded. If you are using WebSockets, it's much more efficient to use encode it in the DeepAffex™ Protobufs.

Each chunk also needs an *action* associated with it which tells the Cloud how to process it. The actions are summarized below:

Action	Description
FIRST::PROCESS	Add chunk as first of measurement and process it (Drop any previously added chunks)
FIRST::IGNORE	Add chunk as first of measurement but *do not* process it (Drop any previously added chunks)
CHUNK::PROCESS	Add chunk to started measurement and process it
CHUNK::IGNORE	Add chunk to started measurement but *do not* process it
LAST::PROCESS	Add chunk as last of measurement, process it and finish measurement
LAST::IGNORE	Add chunk as last of measurement, but *do not* process it and finish measurement

Each chunk also has a `StartTime` and `EndTime` in seconds. The `StartTime` of a chunk cannot be less than `EndTime` of the previous chunk. The difference between `StartTime` and `EndTime` cannot be more than 100 seconds nor less than 5 seconds. Usually, it's best to use the values of `StartTime`, `EndTime` etc. that are returned by the DeepAffex™ Extraction SDK.

Note: This endpoint is subject to request throttling. Users are encouraged not to submit more data than can be obtained in real time. i.e. do not send more than 2 minutes of chunk durations over the course of 2 minutes of real time.

Decoding real-time results

Real-time results that are returned via the `Measurements.Subscribe` endpoint can be decoded using the `decodeMeasurementResult` method of in the DeepAffex™ Extraction SDK.

Next steps

The following chapter has more details about decoding and other aspects of the DeepAffex™ Extraction SDK.

DeepAffex™ Extraction SDK

The DeepAffex™ Extraction SDK is used to extract facial blood-flow information from a sequence of face-tracked images. The SDK is available as a library on most mainstream platforms and is available to download on the DeepAffex™ website. It has bindings to several popular programming languages made possible by its C API which facilitates linking with C Foreign Function Interfaces.

The SDK is canonically documented in the C/C++ documentation.

Installation

C / C++

The C / C++ SDK consists of:

- `libdfx` - a precompiled C library distributed as header files and binaries
- `libdfxcpp` - a C++ wrapper for `libdfx`, distributed as C++ source files and header files

We provide 64-bit binaries for Windows, macOS and Linux. The C++ wrapper layer that sits on top of the C API is a convenient access layer for C++ applications. It requires access to an OpenCV implementation. (The core C library has no external dependencies).

Installing the C++ SDK involves simply copying the headers and libraries locally and configuring your toolchain to point to them.

The README.html in the zip file, contains details on how to use CMake to build the C++ wrappers and also on how to compile an example program. You may choose to forgo CMake and e.g. use Visual Studio directly if you wish.

C / C++ documentation is available online.

.NET

The .NET DFX SDK is distributed a NuGet package targetting .NET Framework 4.6.1. It wraps the 64-bit DFX SDK C dll - `dfx.dll`.

Please refer to the NuGet or Visual Studio Package Manager documentation on how to install is as part of your project.

Python

The Python DFX SDK is distributed as Python Wheels for Windows, macOS and Linux, wrapping the respective 64-bit DFX SDK dynamic link libraries.

Installation is simple via a `pip install` command.

Basic Usage

Here we discuss basic usage of the DeepAffex™ SDK for blood-flow extraction. We use Python here for clarity but the API is the same in all the languages.

DFX Factory

A DFX `Factory` is the primary entrypoint to the SDK. A `Factory` object is constructed by calling a parameterless constructor.

```
factory = dfxsdk.Factory()
```

The next step is to use the SDK ID to obtain study configuration data from a POST call to the Cloud API's `Studies.Sdkconfig` endpoint and to use it to initialize the `Factory`. (This assumes that you have registered, logged in, obtained a token and selected a study as discussed in the previous chapter.)

We pass the SDK ID, study ID and the hash of the current data on hand. The first time we call this endpoint, the hash would be an empty string. If there is updated study configuration data available, we will get at 200 response and the body will contain the base64 encoded study configuration data and its hash. If our hash we sent is up to date, then we will get a 304 response. Please cache the study configuration data and its hash for future use.

```
sdk_id = factory.getSdkId()
study_cfg_data = # Cloud API call with sdk_id, studyid and current_hash
if not factory.initializeStudy(study_cfg_bytes):
    print(f"DFX factory creation failed: {factory.getLastErrorMessage()}")
```

If the initialization had no errors, our next step is to create a DFX `Collector`.

DFX Collector

A `Collector` collects `Frame`s containing `Face`s and produces chunks of data containing blood flow information (`ChunkData`.) To create a collector, we call the `createCollector` method of an initialized `Factory` object.

```
collector = factory.createCollector()
```

The collector uses frame timestamp information to determine when a chunk of data is ready to be sent to the DeepAffex™ Cloud for processsing. Thus, before we can start using it, we have to set some important collector properties - the anticipated framerate, the duration of each chunk of data and the duration of the measurement.

```
collector.setTargetFPS(fps)
collector.setChunkDurationSeconds(chunk_duration_s)
collector.setNumberChunks(number_chunks)  # measurement duration =
chunk_duration_s * number_chunks
```

Next, we create a measurement on the DeepAffex™ Cloud using a POST call on the `Measurements.Create` endpoint.

At this stage, we are ready to start collecting blood flow data. However, a few points are to be noted.

- The minimum chunk duration is 5 seconds. Intermediate results will be available at this interval.
- The chunk duration may not evenly divide the total duration of a measurement e.g. in the case of a video of predetermined length. In that case, we set the number of chunks to be one more than the quotient. For the last chunk, when we have received the last frame, we use the `forceComplete` method of the collector.
- In a live camera measurement, we could delay the start of blood flow data collection, until the person's face is in the frame and certain conditions like lighting etc have been met. This is the DFX Constraints system and is discussed in more detail in the next section.

Assuming we have an image source producing frames and a face tracker that can track each frame and produce facial landmark information, we proceed by calling `startCollection` on the collector object and adding frames to it.

```
collector.startCollection()
```

Adding frames

To add frames to the collector:

1. we first wrap our native image format into a DFX `VideoFrame` object (which tells the collector things like the timestamp of the frame and the channel order of the frame.)

```
dfx_video_frame = dfxsdk.VideoFrame(image,
                                    frame_number,  # Relative frame number
                                    frame_timestamp_ns,  # Relative frame
timestamp in nanoseconds
                                    dfxsdk.ChannelOrder.CHANNEL_ORDER_BGR)
```

2. Then, we create a DFX `Frame` object by passing our `VideoFrame` to the collector.

```
dfx_frame = collector.createFrame(dfx_video_frame)
```

3. Then, we add the DFX `Face`s containing face tracking information to the `Frame`.

```
for dfx_face in dfx_faces:
    dfx_frame.addFace(face)
```

4. Finally, we create regions (from where the facial bloodflow information will be extracted) and we extract the information. We also check and see if a chunk of data is ready and whether the measurement has ended.

```
collector.defineRegions(dfx_frame)
result = collector.extractChannels(dfx_frame)

if result == dfxsdk.CollectorState.CHUNKREADY or \
   result == dfxsdk.CollectorState.COMPLETED:
    chunk_data = collector.getChunkData()
    if chunk_data is not None:
        chunk = chunk_data.getChunkPayload()
        # Send the chunk to the DeepAffex™ Cloud
    if result == dfxsdk.CollectorState.COMPLETED:
        # Exit our image collection loop
```

Note: `getChunkData` may return a `None` (or a `nullptr` in C++). This will happen if the SDK wasn't able to extract enough bloodflow information from the frames that were passed in.

Decoding results

The `Collector` also has a `decodeMeasurementResult` function that can decode the encoded binary results that are received on the Websocket-only Measurements.Subscribe endpoint.

```
decoded_result = decodeMeasurementResult(payload)
```

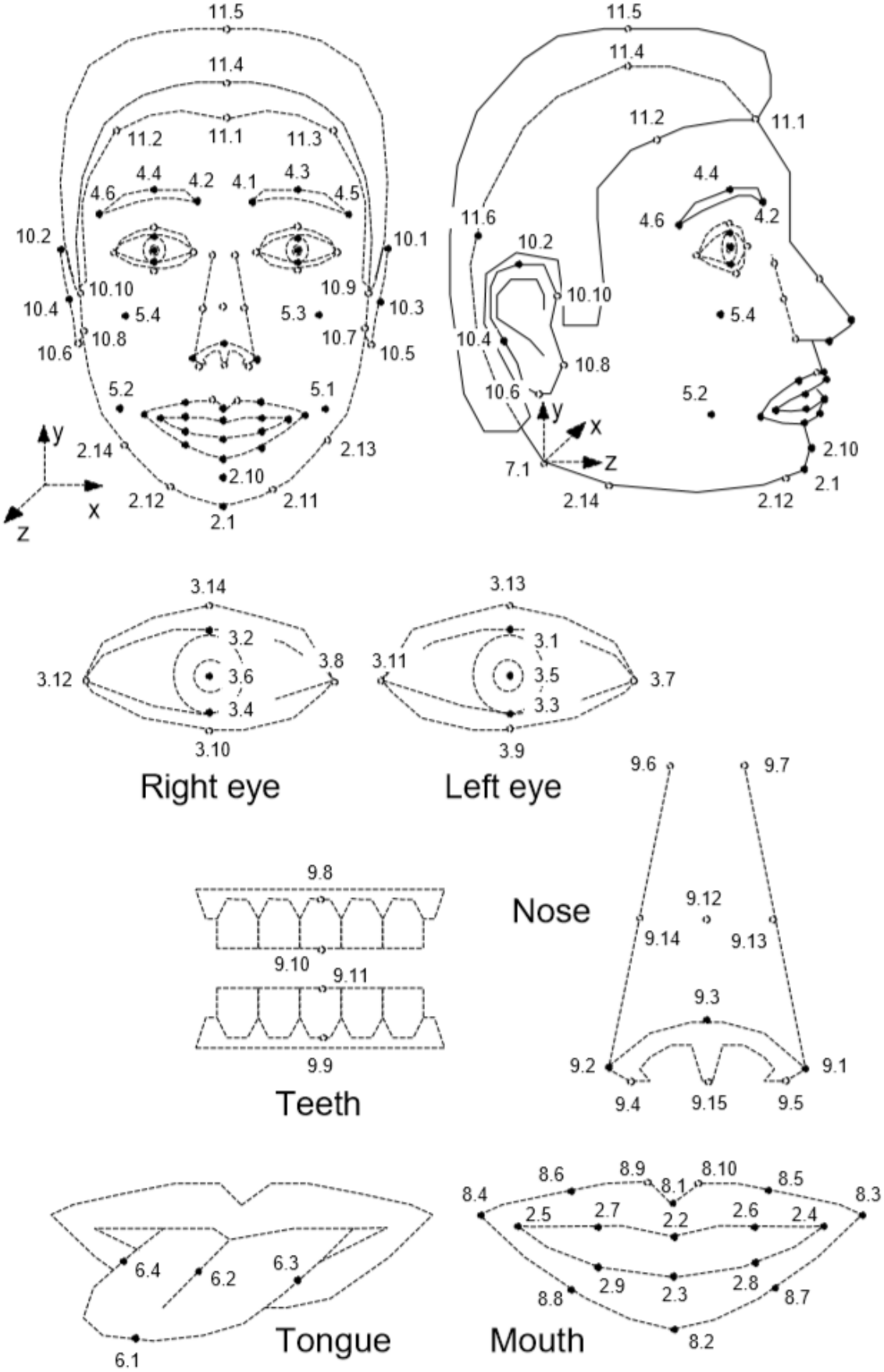
Some details about DFX `Face`'s follow in the next section

Adapting Face Trackers

The DFX SDK does not provide any built-in face pose estimation. Face tracking and pose estimation is available as a commodity with implementations like Dlib, Visage etc.

MPEG-4 Facial Data Points

Whatever the output of the face tracking engine, it will need to be mapped into standard MPEG-4 Facial Data Points before being inserted into a DFX `Frame` as a DFX `Face` structure.



- Feature points affected by FAPs
- Other feature points

Please see the `_dlib2mpeg4` structure in `dlib_tracker.py` in the `dfxdemo` as an example of such a mapping for the `Dlib` face tracker. NuraLogix™ will be happy to assist you in this process and we already have similar mappings for several common face tracking engines.

Required Landmarks

The DFX `Collector` has a method called `getRequiredPosePointIDs` which will return a list of MPEG-4 Facial Data Point names that need to be added to the `Face` structure for the blood-flow extraction to work correctly. This list will change base on the study configuration data used to initialize the DFX `Factory`.

However, there are currently 15 core facial landmarks used by the majority of DFX Studies. These landmarks and other details are discussed thoroughly in the next chapter.

DFX Face

A DFX `Face` structure consists of:

- `id` - unique to one person in a frame. The person's name, a GUID or even a counter will suffice
- `faceRect` - the bounding rectangle of the face in the frame
- `poseValid` - true if the posePoints are valid
- `detected` - true if the face was detected in this frame, false if the face-tracking information is cached from an older frame and was used to estimate landmark points on this frame.
- `posePoints` - a map of MPEG-4 point names and DFX `PosePoint`s
- `attributes` - a map of additional face attributes like `yaw`, `pitch` etc.

A DFX `PosePoint` structure consists of:

- `x` - the X location on the frame
- `y` - the Y location on the frame
- `z` - the Z location on the frame (reserved for future use)
- `valid` - true if this point was valid in this frame
- `estimated` - false if this point was returned by the face tracker, true if it was estimated by interpolating from points that were returned by the face tracker
- `quality` - the tracking quality (or probability) of this point (between 0 and 1)

Face-tracking strategies

If the face tracker that you have selected cannot keep pace with the frame rate of the image source, then you can run the face tracker on a best-effort basis on a background task and use it's most recently produced results (within reason.) Please be sure to set the `detected` and `estimated` fields correctly when using cached data.

Currently, the DFX solution can better tolerate some inaccuracies in `PosePoint` locations better than dropped frames. (This may change in the future and a different face tracker strategy may be more appropriate.)

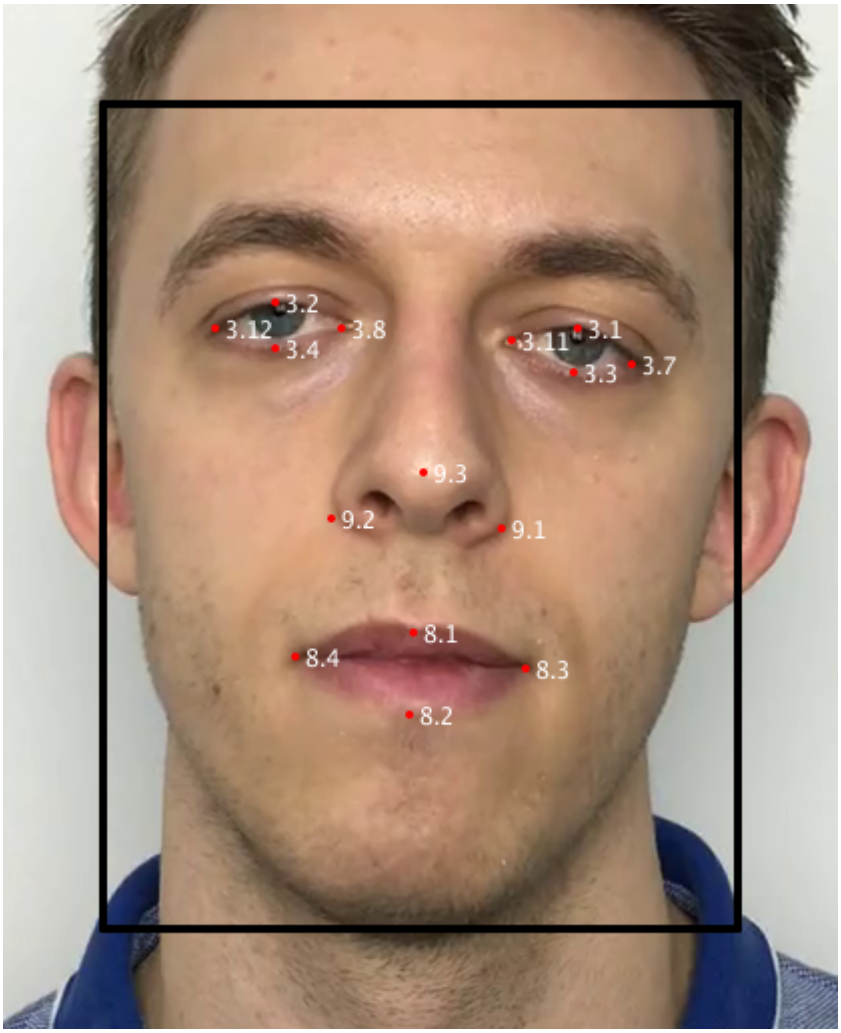
Required Facial Landmarks

Each DFX Study potentially requires a unique set of facial landmarks. However, there are currently 15 core facial landmarks used by the majority of studies. These points provide details to the DFX `Collector` from which it is able to identify the regions of interest for blood-flow extraction.

In addition, there are 44 visual facial landmarks which are used for animation and rendering of which do not affect the measurement quality.

DFX Core Measurement Points

The 15 core facial data points are identified in this image by the red dot at the location and with the corresponding white label. The black box is the detected face rectangle provided to the SDK along with the face points.

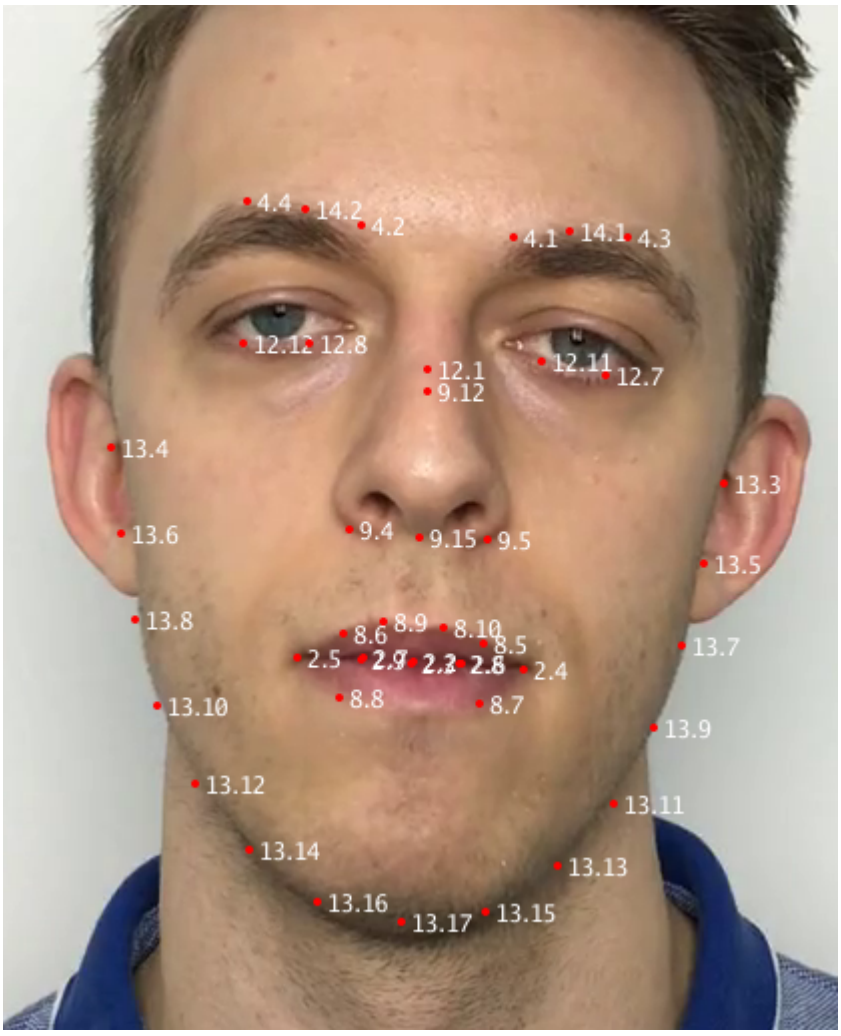


In addition to providing the black face bounding box, the 15 required point names are:

3.1, 3.2, 3.3, 3.4, 3.7, 3.8, 3.11, 3.12, 8.1, 8.2, 8.3, 8.4, 9.1, 9.2, 9.3

DFX Visual Animation Points

There are 44 animation points which are used to construct the visuals which end users typical see along the contour of the face. These points do not need to be tracked (or mapped) as accurately as the core measurement points as they are only for aesthetic purposes. It is actually difficult for most face engines to accurately track the outer profile of a face and so these points are anticipated to have much more inaccuracy in general. You can see in the positioning of the group 13 series below how they have drifted from the edge of the face in the following diagram.



The 44 points used for visual animation include:

2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 4.1, 4.2, 4.3, 4.4, 8.10, 8.5, 8.6, 8.7, 8.8, 8.9, 9.12, 9.15, 9.4, 9.5, 12.1, 12.11, 12.12, 12.7, 12.8, 13.10, 13.11, 13.12, 13.13, 13.14, 13.15, 13.16, 13.17, 13.3, 13.4, 13.5, 13.6, 13.7, 13.8, 13.9, 14.1, 14.2

Raw Data

In order to help facilitate mapping a face tracker to the MPEG-4 points that the DFX SDK leverages, the raw data for the annotated image is:

- raw_facial_image.png
- raw_facial_data.csv

The first line in the CSV file is a header describing the data. The next two lines are for the top left corner of the bounding box (`box.tl`), and the bottom right corner of the bounding box (`box.br`), respectively. The rest of the data lines are for 174 landmarks points provided by the tracker we used to produce this data.

Constraints

The DeepAffex Cloud will occasionally fail to return a result successfully. This usually occurs when the signal to noise ratio of the extracted blood-flow signal isn't high enough or when there are other issues with it.

While the user cannot do much about this for blood-flow extracted from videos, if a measurement is being made live using a camera, your application can provide actionable feedback to the user based on the SDK Constraints system which may increase the chances of a successful result. You can also cancel a failing measurement early without waiting for a round trip over the internet.

Constraints demo using `dfxdemo`

`dfxdemo`, introduced earlier, has the ability to make measurements using your computer's webcam and provide actionable feedback using the Constraints system while you do so. Assuming you have completed all the prerequisite steps needed to make measurements (i.e. registered your license, logged in as a user and selected a study,) you can run the following command:

```
python dfxdemo.py measure make_camera
```

Note: If your computer has multiple webcams and you want to select a different one, you can use the `--camera <index>` argument. You can pass `--help` to get a full list of options.

Once the demo starts, you may see multiple feedback messages in red, asking you, for example, to look straight at the camera or to hold still. Once all the constraints are met, you can press `s` to start the measurement. You may also see warnings during the measurement and if you violate constraints too much during the measurement, it may fail.

Checking constraints

Programmatically, you need to call `checkConstraints` on every `Frame` - typically, *before* starting a collection.

```
dfx_frame = collector.createFrame(dfx_video_frame)
for dfx_face in dfx_faces:
    dfx_frame.addFace(face)
result, details = collector.checkConstraints(dfx_frame)
```

The `result` return value is the aggregate result of the constraint check and can be either `GOOD`, `WARN` or `ERROR`.

- `GOOD` means that the constraints are all met and collection can proceed. If it hasn't started yet you could choose to automatically start it or allow the user to start it using an input. (Recall that a measurement is started by calling `startCollection`)
- `WARN` means that constraints are currently violated but the user has still has a chance to change conditions and change things back to `GOOD` and perhaps still have a successful measurement. Feedback should be provided to the user.
- `ERROR` means that the collection has failed because of too many constraint violations. Obviously, `ERROR` can only be returned after the collection has started. On an `ERROR`, the application is expected to call `cancelCollection` to terminate the ongoing measurement and `resetCollection` to reset the internal collector state.

On a result of `WARN` and `ERROR` the `details` return value contains a map of reason strings and `result` values. The reason strings correspond to each possible constraint that is violated with a `WARN` OR `ERROR`.

The possible reason strings and their meanings are listed below:

Reason String	Meaning	Possible user feedback
FaceNone	No faces detected in the frame	Move face into frame
FaceOffTarget	User's face is not in the constraints target region	Move face into target region
FaceDirection	User is not looking at camera	Look straight at the camera
FaceFar	User's face is too far from camera	Move closer to the camera
FaceMovement	User is moving too much	Hold still
ImageBright	Image is too bright	Make the image darker
ImageDark	Image is too dark	Make the image brighter
ImageQuality	Image quality is not good	Improve image quality
ImageBackLit	User's face is backlit	Remove illumination from behind the face
LowFps	Frame rate is too low	Improve frame rate

Clearly, feedback to the user can and should be customized for your specific application. The `Image` and `LowFps` reasons in particular can be dealt with using application specific camera settings and controls. In addition to the above, you may get `ImageEmpty` (an empty image was passed) and `CameraMovement` (capture device was moving during capture) which will not be discussed in this guide.

Configuring constraints

The constraints that will be checked by the DFX `Collector` are configurable by the developer. You can query the currently set constraints by calling `collector.getConstraintsConfig("json")` and modify the values by calling `collector.setConstraintsConfig("json", json_string)` - the return value and the modified values are expected to be JSON strings which will contain a simple JSON dictionary. Typically, you would get the existing constraints, modify what you need to and set them again.

```
constraints_cfg_str = collector.getConstraintsConfig("json")
# Parse the json, modify the values as needed
modified_constraints_cfg_str = parse_and_modify(constraints_cfg_str)
collector.setConstraintsConfig("json", modified_constraints_cfg_str)
```

The following constraints can be adjusted:

Minimum frame rate

The capture frame rate in frames/second has to exceed `minimumFps` (default: 27). This check can be disabled using `checkMinFps`.

Face position and presence

The face has to be present within a target rectangle within the frame. (The rectangle is specified as a percent of the frame size using `boxCenterX_pct`, `boxCenterY_pct`, `boxWidth_pct` and `boxHeight_pct`) (default: 50, 50, 50, 99). These defaults are optimized for the landscape orientation. This check can be disabled using `checkCentered`.

Face distance from camera

The face has to be close enough to the camera. Minimum inter-pupillary distance in pixels (`minInterPupilDist_px`) is used a proxy for distance (default: 60.0). This check can be disable using `checkDistance`. (This check assumes that you haven't disabled the face rotation constraint described below.)

Image brightness and backlighting

The rectangle containing the face has to be brighter than `threshDark` (default: 48) and darker then `threshBright`. (default: 225). This check can be disabled using `checkLighting`.

If the region around the face, created by growing the face rectangle by `backLightSearchMult` (default: 2.0) and cropping at the bottom of the face rectangle, has more then `backLightMaxPixels_pct` percentage (default: 30%) of its pixels brighter then `backLightThresh` (default: 240) then the backlight constraint is violated. This check can be disabled using `checkBackLight`.

Face rotation

The user has to look at the camera. The maximum allowed left-right face rotation is `maxFaceRotLR_deg` degrees (default: 12.5) and max allowed up-down face rotation is `maxFaceRotUD_deg` degrees. (default: 25). This check can be disabled using `checkFaceDirection`.

Face movement

The nose position is tracked in time (for the last `movementWindow_ms` milliseconds) (default: 2000) and if it exceeds `maxMovement_mm` millimeters (default: 10), then a warning is issued. This check can be disabled using `checkMovement`.

Eyebrow movement

The perpendicular distance from the eyebrow midpoint to the line joining the two lateral canthuses is calculated in mm for both eyebrows and tracked. If the range of the larger tracked value exceeds `maxEyebrowMovement_mm` millimeters (default: 3), then a warning is issued. This check can be disabled using `checkEyebrowMovement`

NOTE: The face movement and eyebrow movement constraints are only checked if a measurement is active.

We recommend using your judgement to modify and/or disable constraints to get the best results for your particular use case. Testing with a wide variety of users and in a wide range of environments is very helpful!

DeepAffex™ Dashboard

The DeepAffex™ Dashboard is a web-based management tool primarily intended for developers and system administrators. This allows authorized users to monitor an organization's private account on the DeepAffex™ cloud.

The Dashboard applies Role-Based-Access authorization criteria to provide secure management for an Organization's collection of raw data, measurement results, logs, study configurations, participant information, user accounts, groups and policies, storage and utilization levels and other relevant aspects.

Accessing the Dashboard

The DeepAffex™ Dashboard is available on the DeepAffex™ website. Please note that by default you will be automatically routed the server nearest to your geographical location. If you wish to connect to a specific region, please contact NuraLogix™ for the correct URL for your geography.

Role Based Access

The DFX Dashboard applies Role Based Access (RBA) authorization criteria to provide secure management for an Organization's collection of raw data, measurement results, logs, study configurations, subject/participant information, user accounts, groups and policies, storage and utilization levels and other relevant aspects.

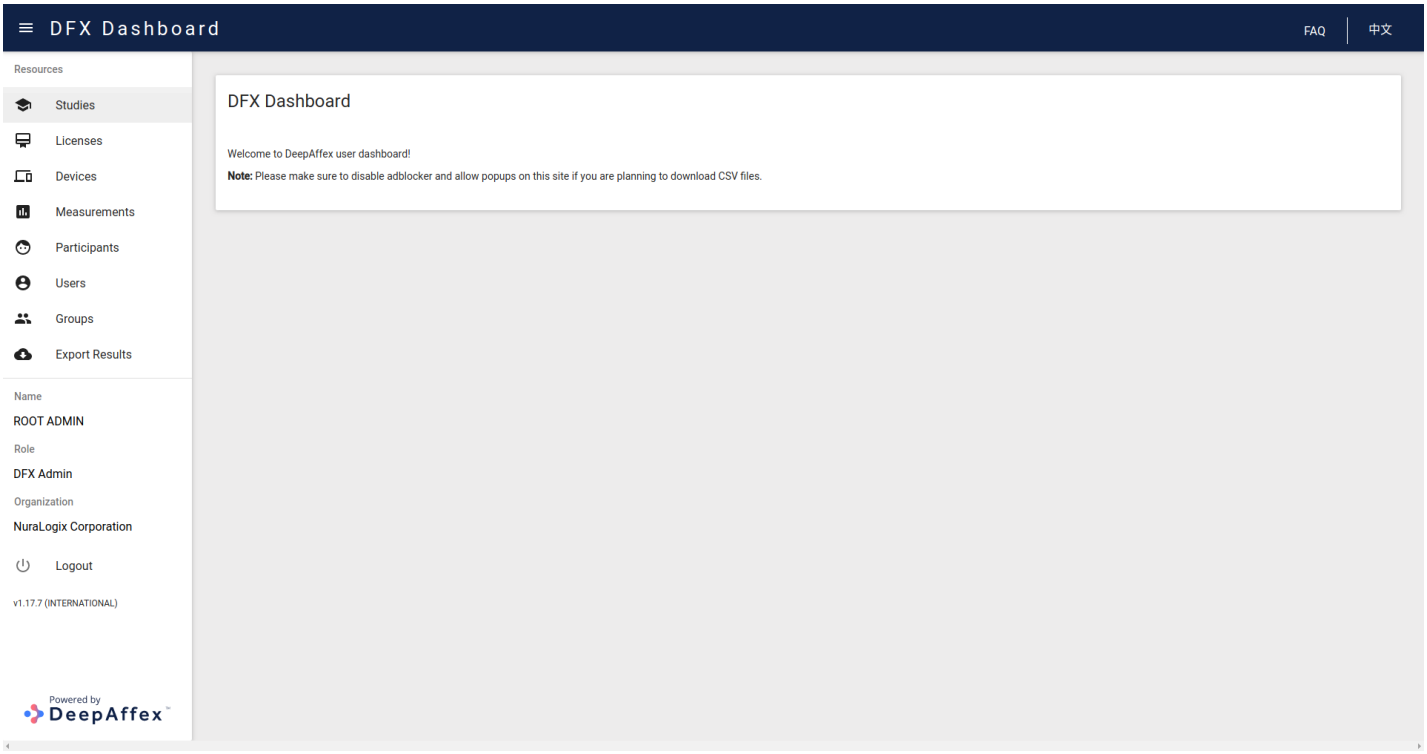
A full list of permissions by user role is shown in the table below.

	Org Admin	DFX Lead	DFX Researcher	DFX Operator	Anura Application User
Studies	✓	✓	✓	✓	✓
Licenses	✓	✓	✓	✓	✓
Devices	✓	✓	✓	-	-
Measurements	✓	✓	✓	✓	✓
Participants	✓	✓	✓	✓	✓
Users	✓	✓	-	-	-
Groups	✓	✓	-	-	-
Export Results	✓	✓	-	-	-

A partial walkthrough using an actual operational DFX account follows.

Navigation

All features and capabilities covered by the Dashboard are shown on a *navigation bar* that is always accessible on the left.

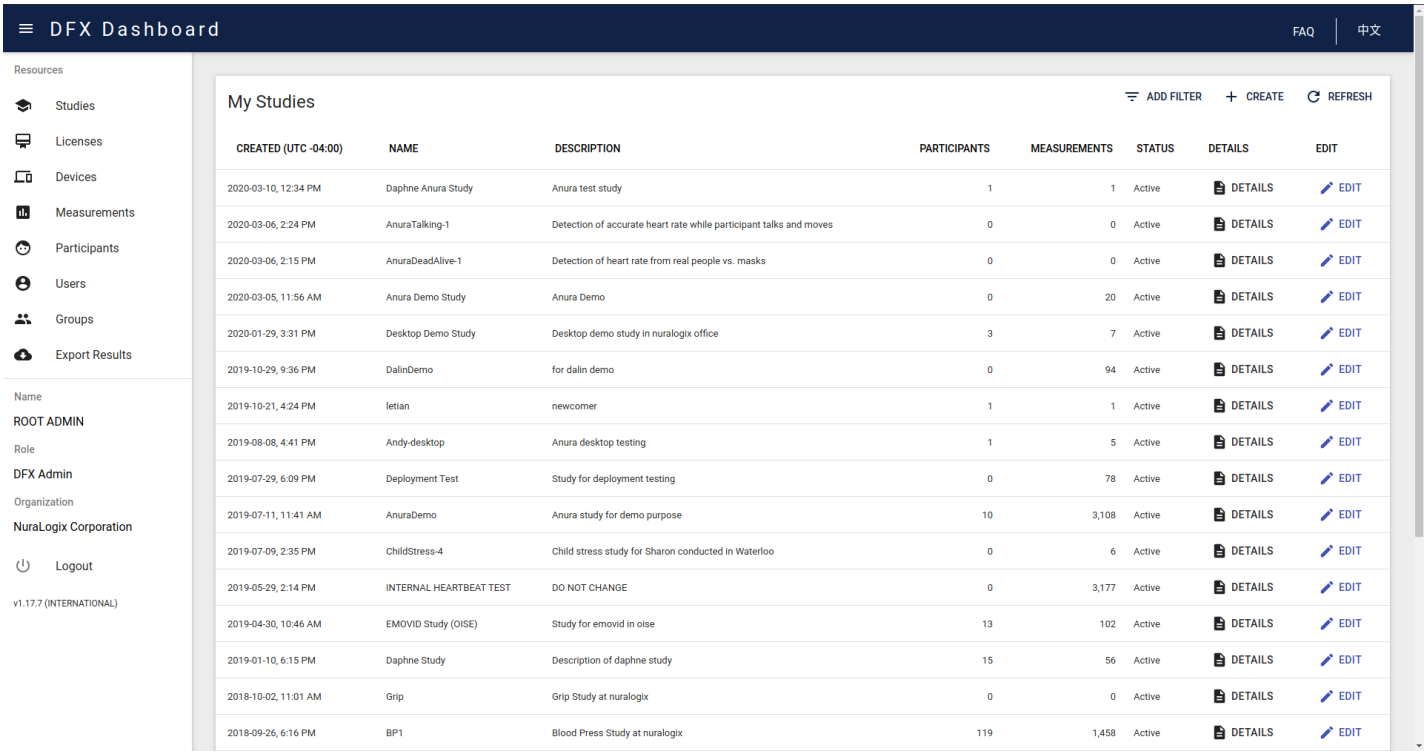


Under all the navigation buttons, the navigation bar will list your name, Role and Organization as well as the version of the Dashboard and server region in use.

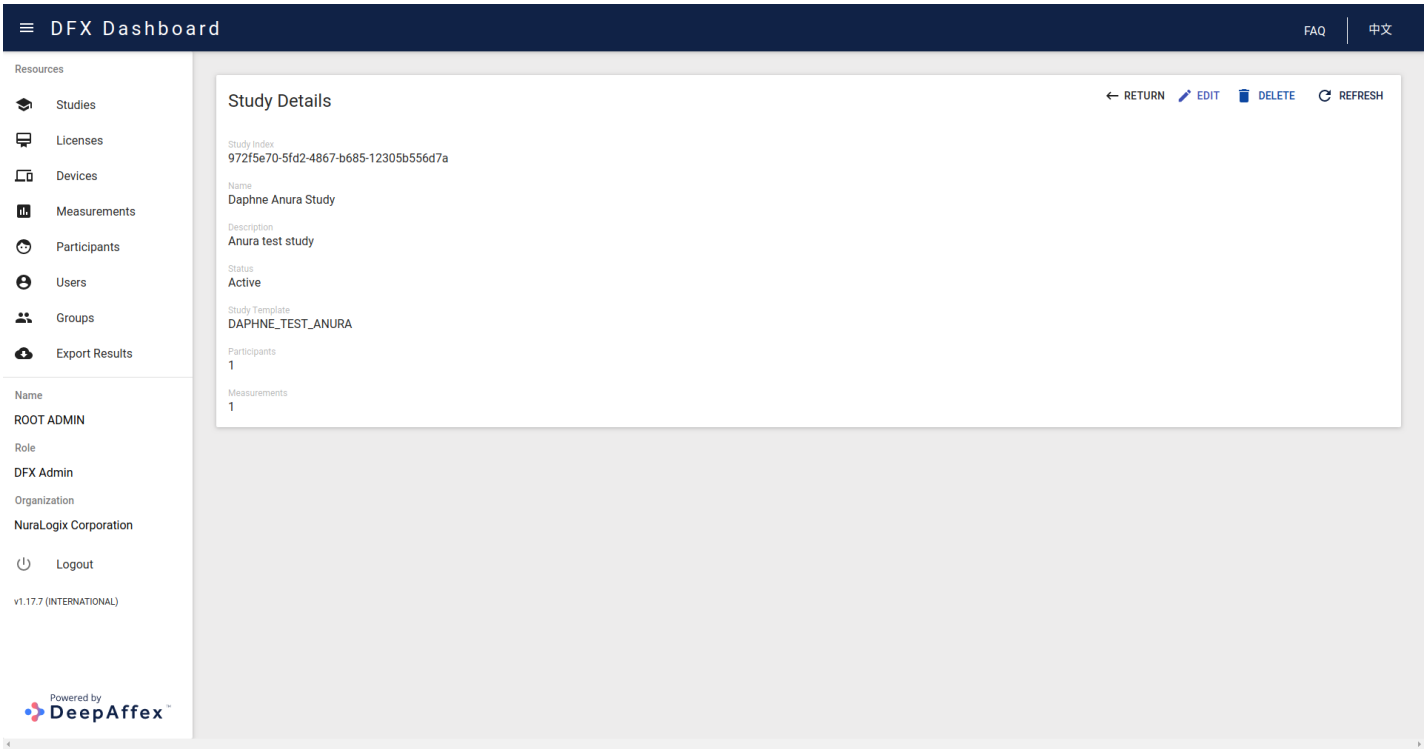
Studies

This section provides access to the parameters for the DFX Studies for which participants are providing measurements under the Organization's account.

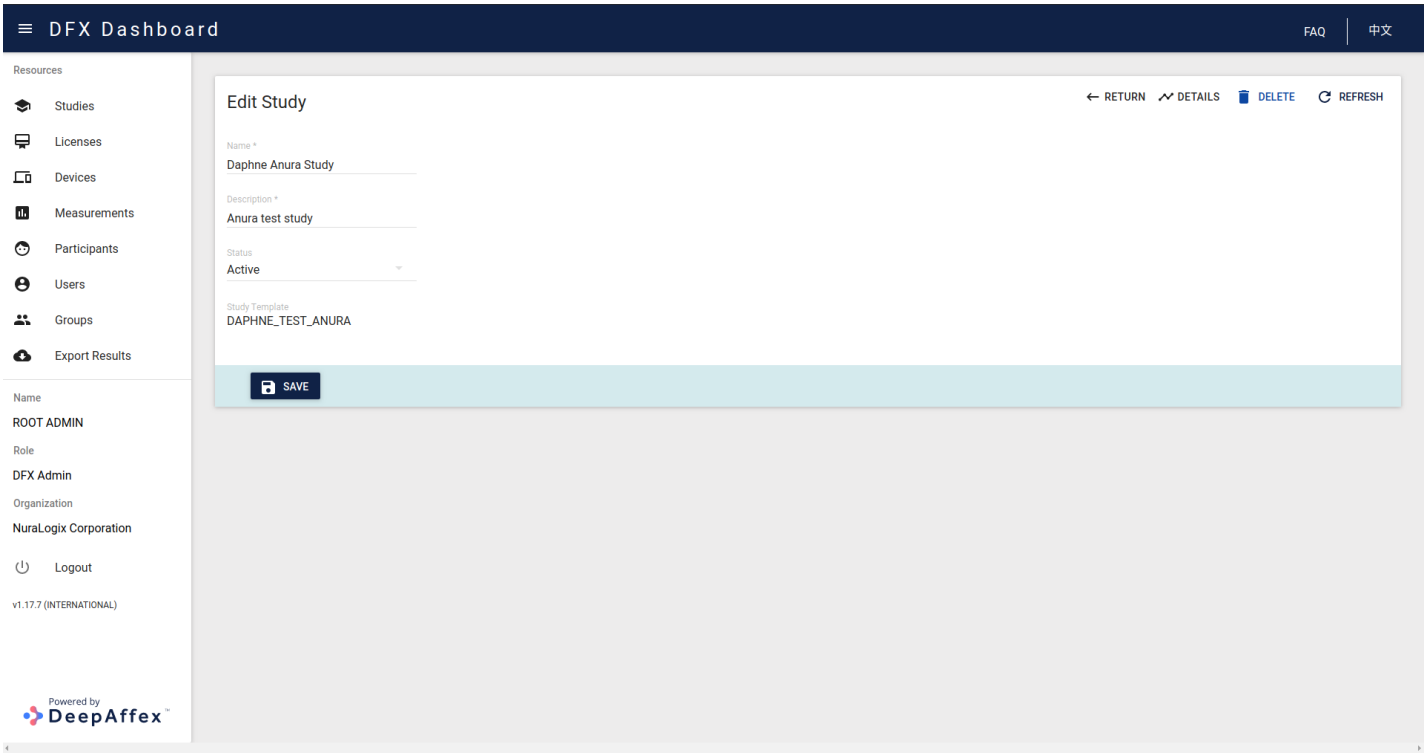
You can sort and filter Studies by date and names.



You can click on a Study's Details button to bring up its details.



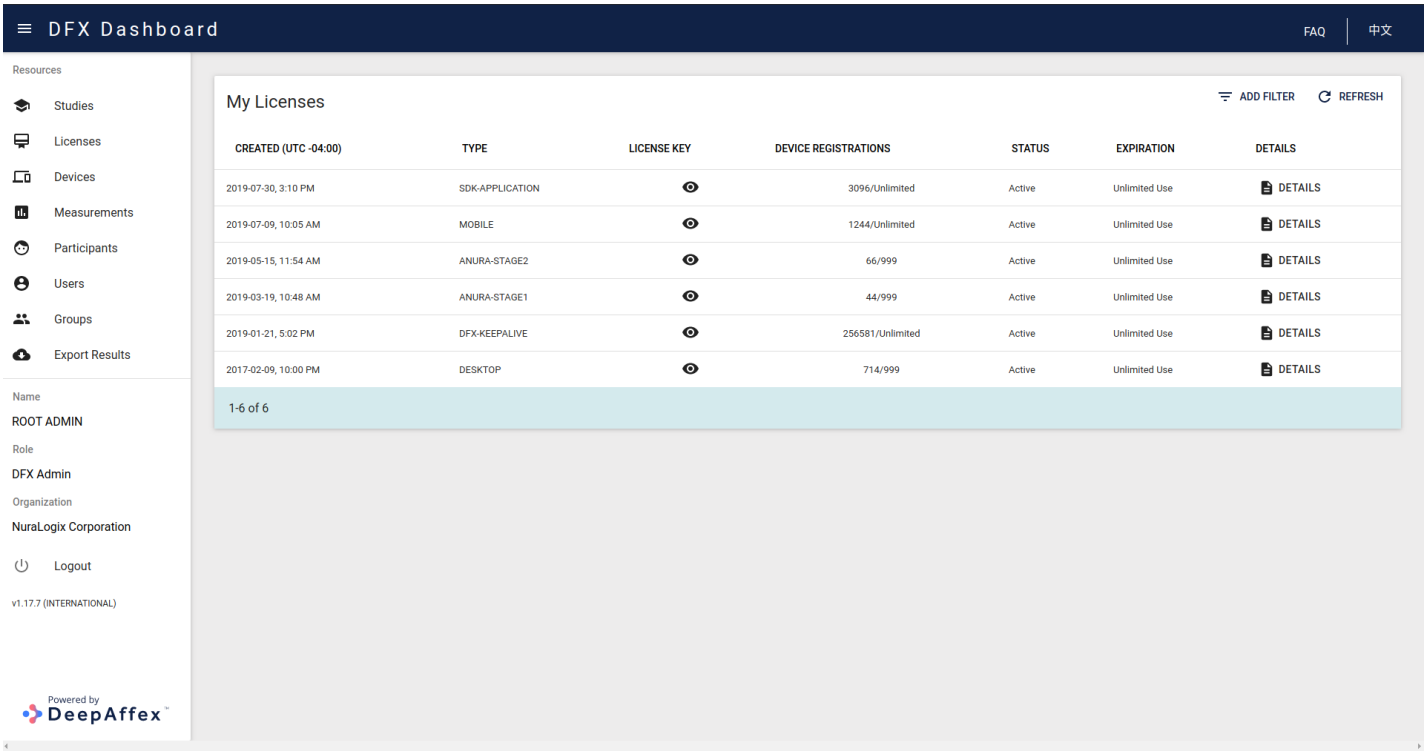
If your role is DFX Lead or higher, you will be able to edit Studies.



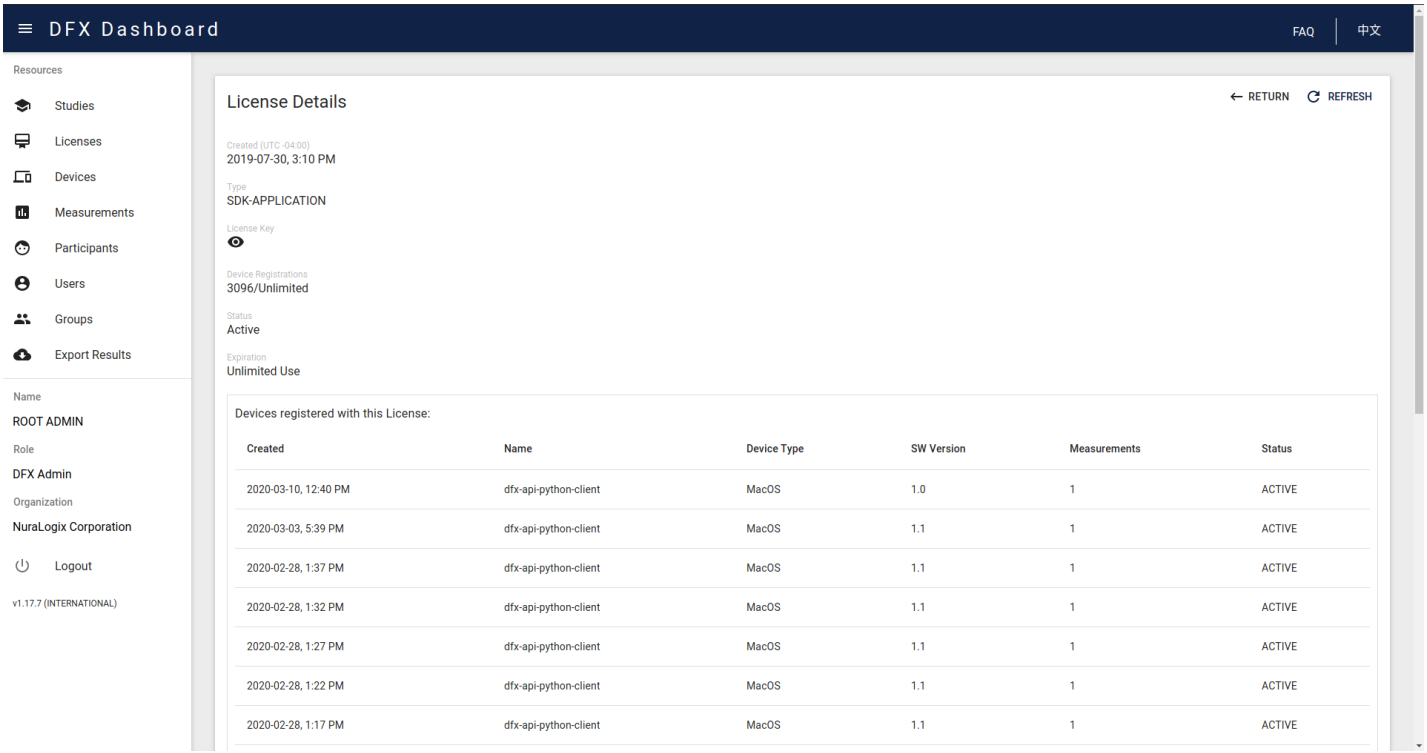
Licenses

In this section, you can review licenses that have been issued to your Organization.

You can sort and filter Licenses by date, type or current status.

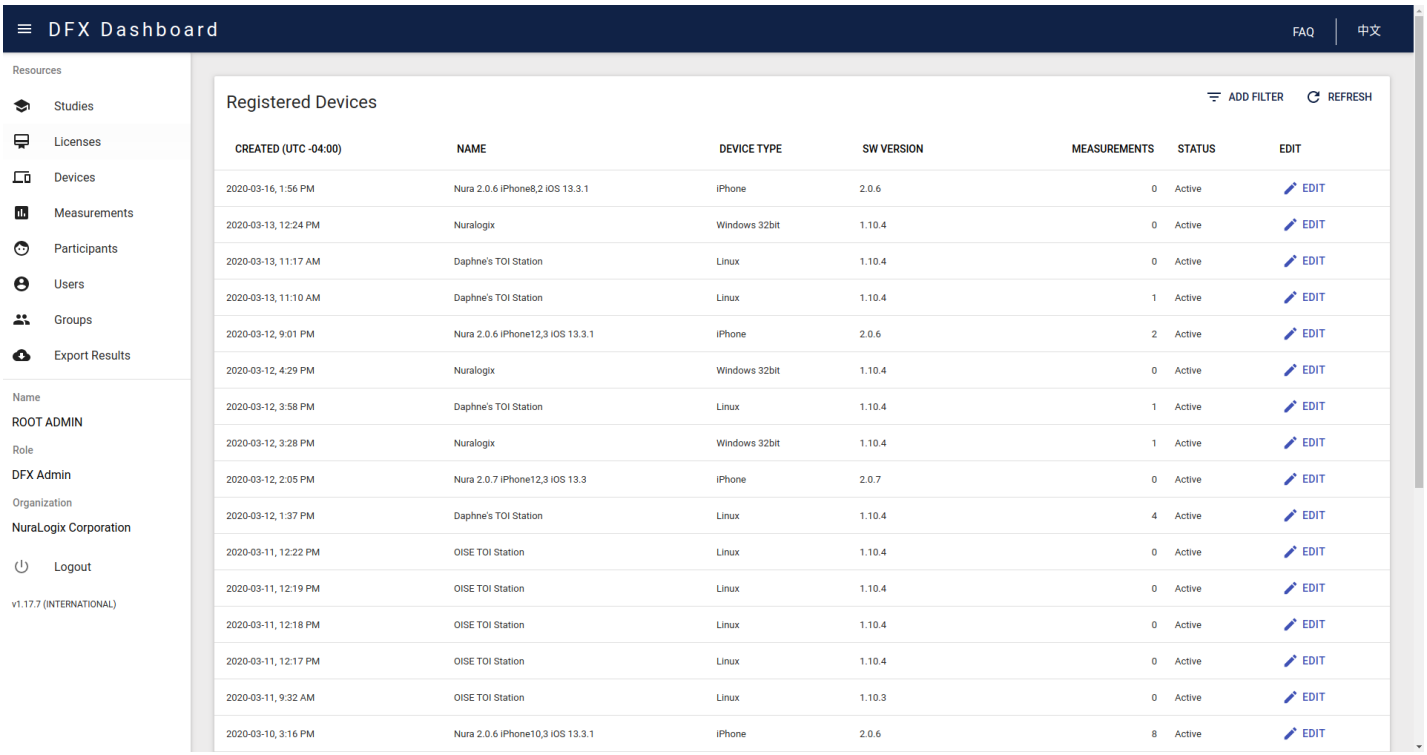


You can click on the Details button to bring up license details including a list of the devices registered with that license.


DFX Dashboard
FAQ
中文
Resources
Studies
Licenses
Devices
Measurements
Participants
Users
Groups
Export Results
Name
ROOT ADMIN
Role
DFX Admin
Organization
NuraLogix Corporation
Logout
v1.17.7 (INTERNATIONAL)
License Details
RETURN
REFRESH
Created (UTC -04:00)
2019-07-30, 3:10 PM
Type
SDK-APPLICATION
License Key
Device Registrations
3096/Unlimited
Status
Active
Expiration
Unlimited Use
Devices registered with this License:
Created
Name
Device Type
SW Version
Measurements
Status
2020-03-10, 12:40 PM dfx-api-python-client MacOS 1.0 1 ACTIVE
2020-03-03, 5:39 PM dfx-api-python-client MacOS 1.1 1 ACTIVE
2020-02-28, 1:37 PM dfx-api-python-client MacOS 1.1 1 ACTIVE
2020-02-28, 1:32 PM dfx-api-python-client MacOS 1.1 1 ACTIVE
2020-02-28, 1:27 PM dfx-api-python-client MacOS 1.1 1 ACTIVE
2020-02-28, 1:22 PM dfx-api-python-client MacOS 1.1 1 ACTIVE
2020-02-28, 1:17 PM dfx-api-python-client MacOS 1.1 1 ACTIVE

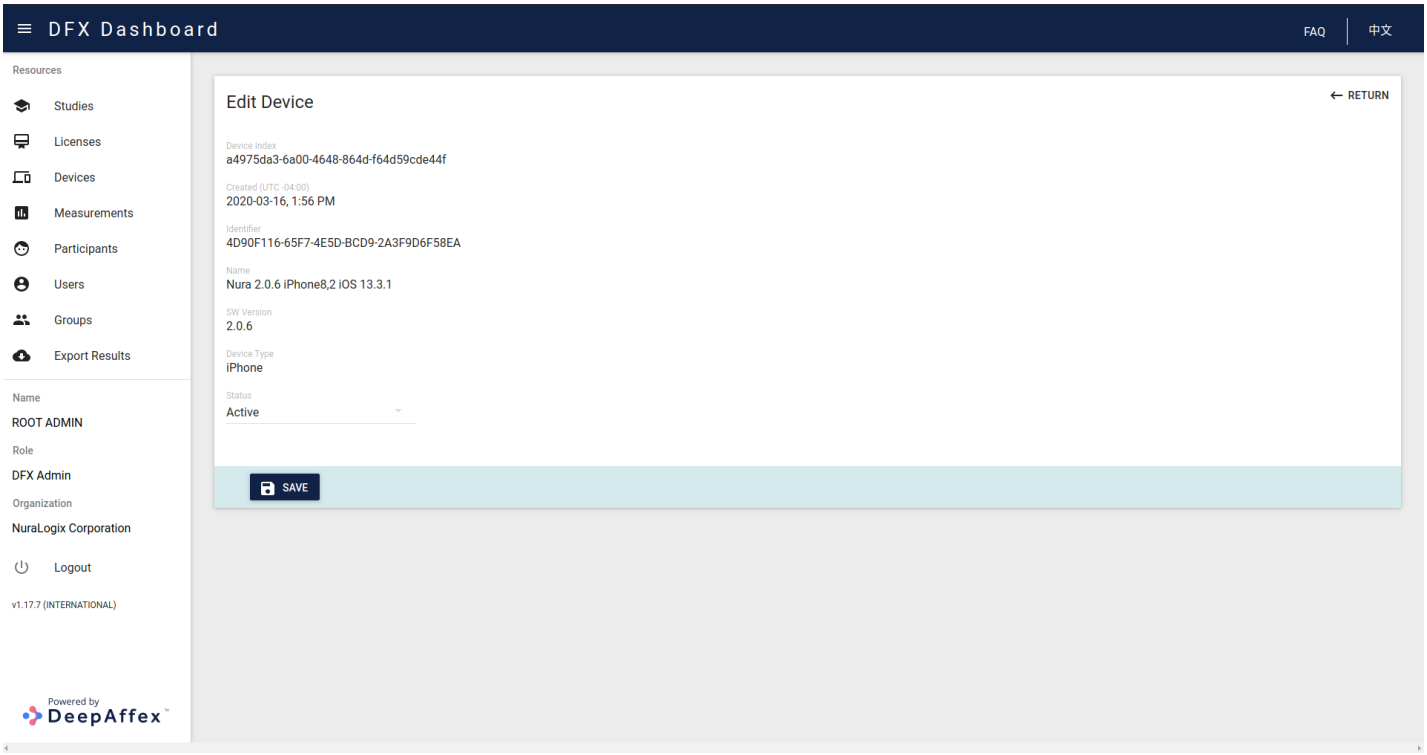
Devices

In this section you can review attributes for devices that are registered to your Organization.



If your role is DFX Researcher or higher, you can view details of individual Devices.

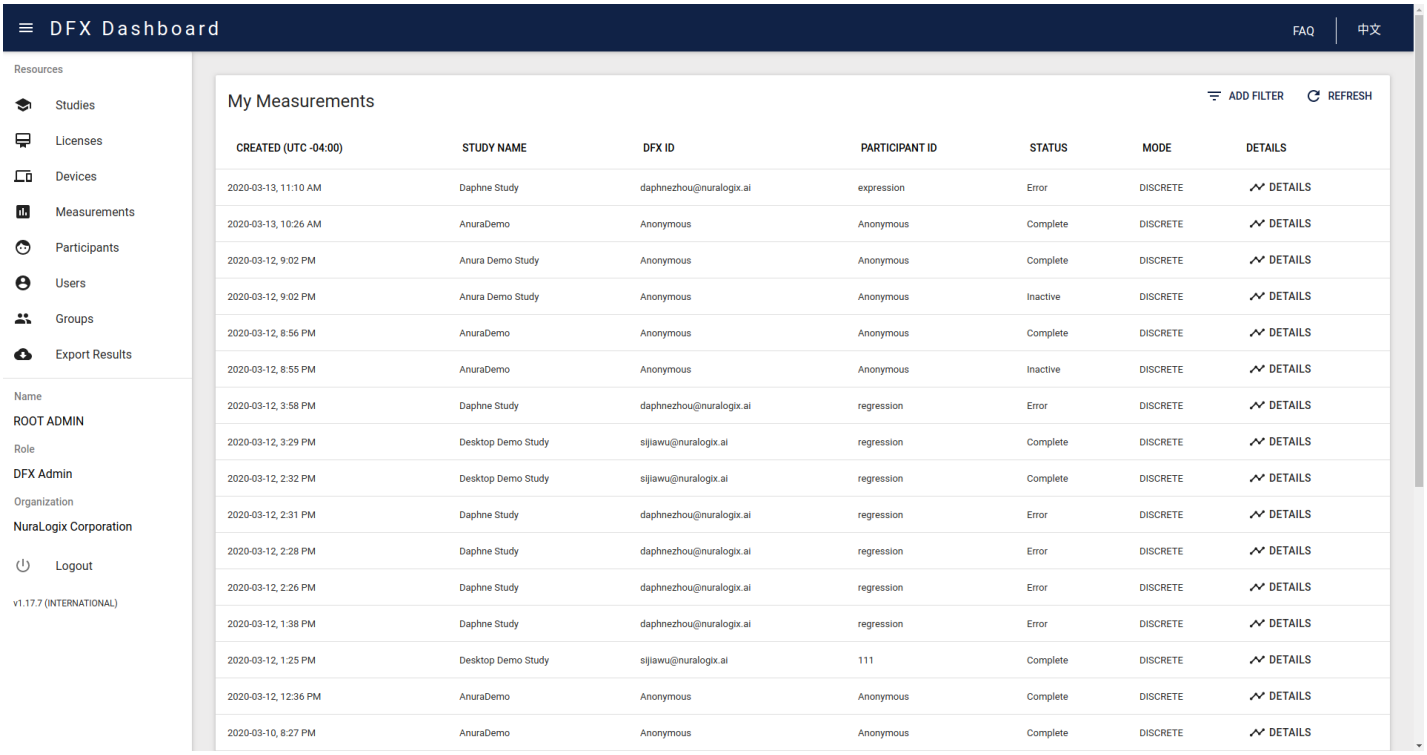
If your role is DFX Lead and higher, you can activate and decommission any device. You can also sort and filter Devices by date, type, version or name.


DFX Dashboard
FAQ
中文
Resources
Studies
Licenses
Devices
Measurements
Participants
Users
Groups
Export Results
Name
ROOT ADMIN
Role
DFX Admin
Organization
NuraLogix Corporation
Logout
v1.17.7 (INTERNATIONAL)
Powered by DeepAffex
Edit Device
← RETURN
Device Index
a4975da3-6a00-4648-864d-f64d59cde44f
Created (UTC -04:00)
2020-03-16, 1:56 PM
Identifier
4D90F116-65F7-4E5D-BCD9-2A3F9D6F58EA
Name
Nura 2.0.6 iPhone8,2 iOS 13.3.1
SW Version
2.0.6
Device Type
iPhone
Status
Active
SAVE

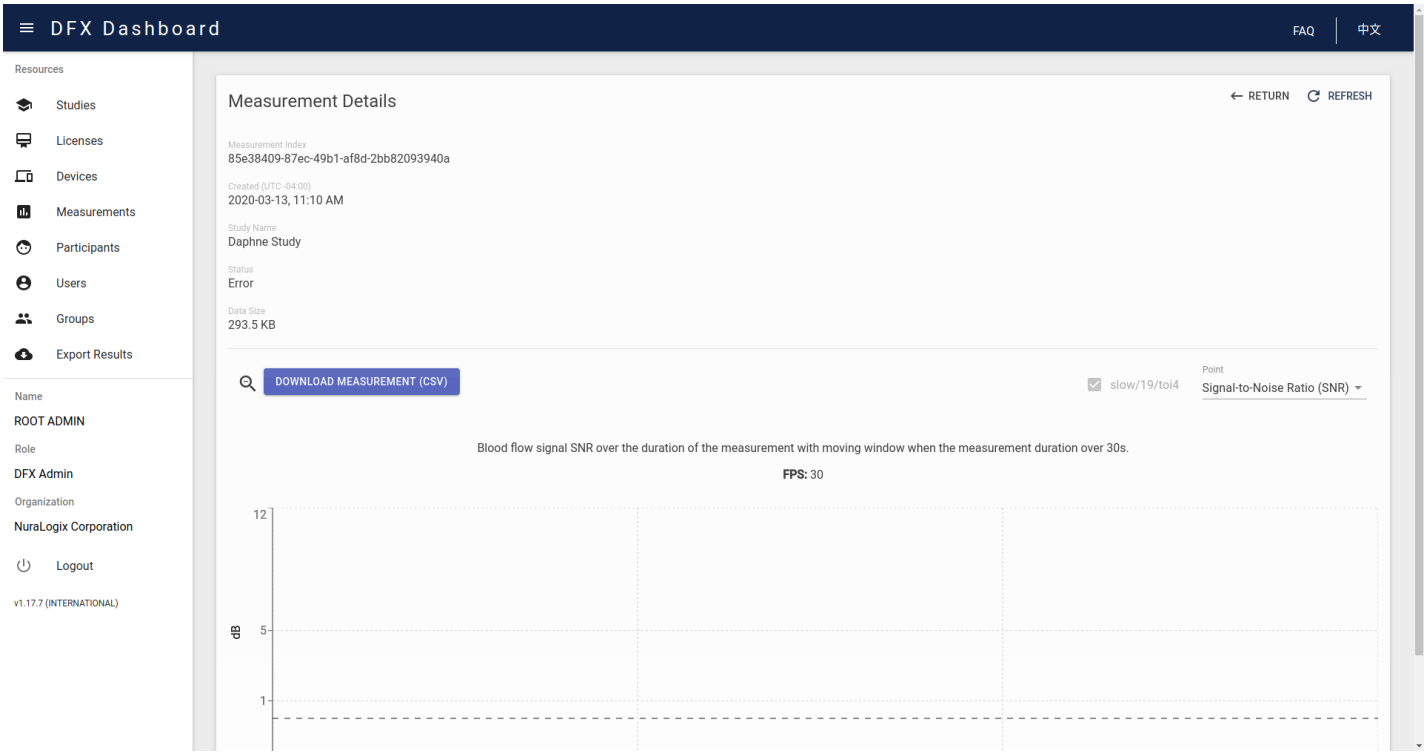
Measurements

In this section, you can review up-to-date measurement results accumulated under your Organization.

You can sort and filter Measurements by date, DFX ID, participant, status, mode or study names.

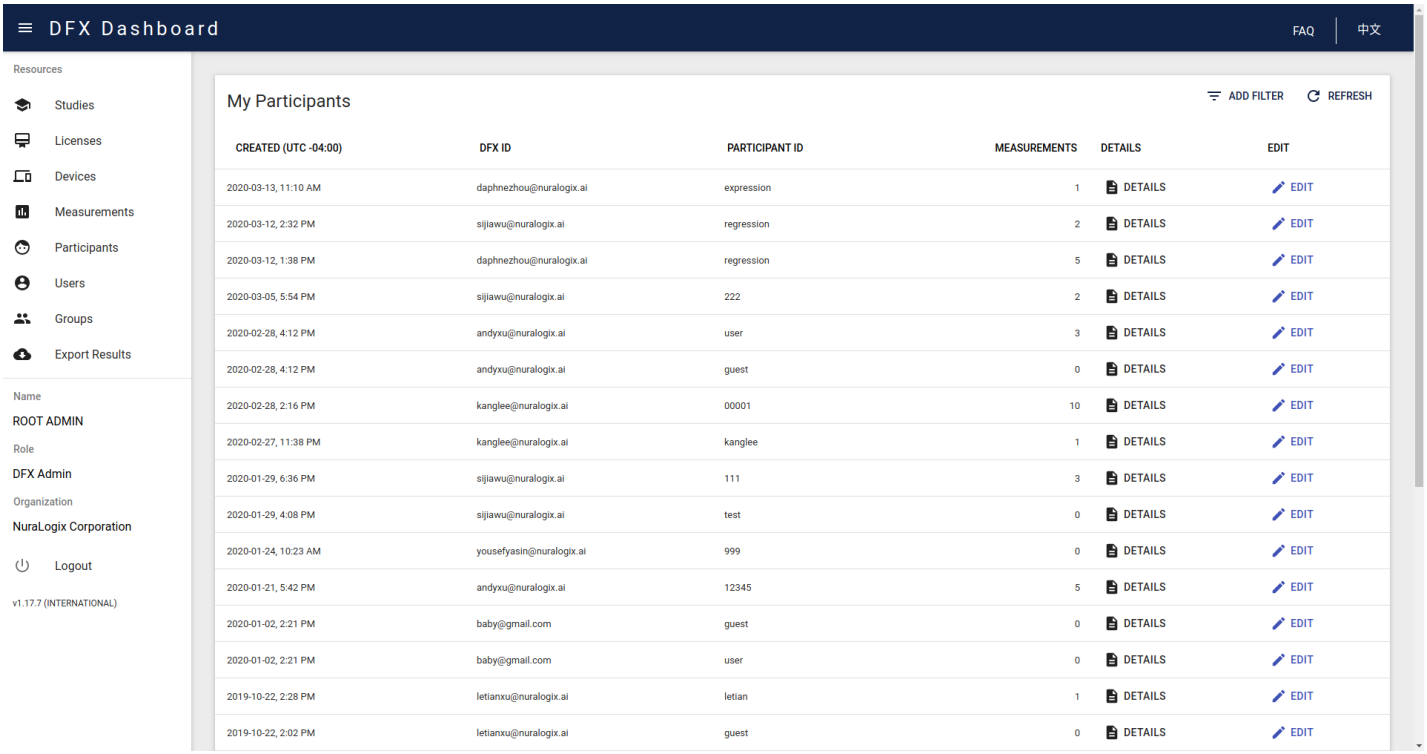


Measurement results can be also downloaded as CSV files from an individual Measurement's Details page. (For bulk export, please see the Export Results section.)

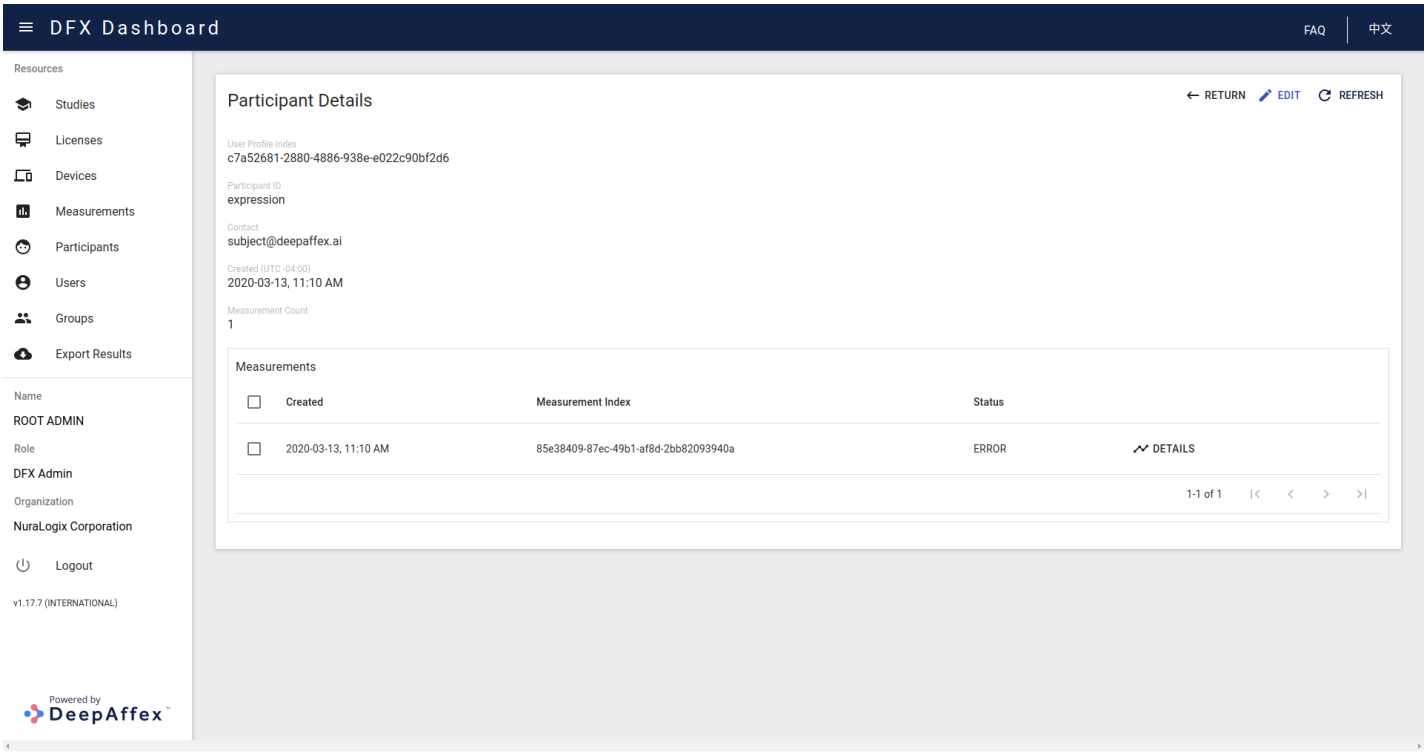

DFX Dashboard
FAQ
中文
Resources
Studies
Licenses
Devices
Measurements
Participants
Users
Groups
Export Results
Name
ROOT ADMIN
Role
DFX Admin
Organization
NuraLogix Corporation
Logout
v1.17.7 (INTERNATIONAL)
Measurement Details
RETURN
REFRESH
Measurement Index
85e38409-87ec-49b1-af8d-2bb82093940a
Created (UTC -04:00)
2020-03-13, 11:10 AM
Study Name
Daphne Study
Status
Error
Data Size
293.5 KB
DOWNLOAD MEASUREMENT (CSV)
slow/19/toi4
Point
Signal-to-Noise Ratio (SNR)
Blood flow signal SNR over the duration of the measurement with moving window when the measurement duration over 30s.
FPS: 30
12
5
1
dB

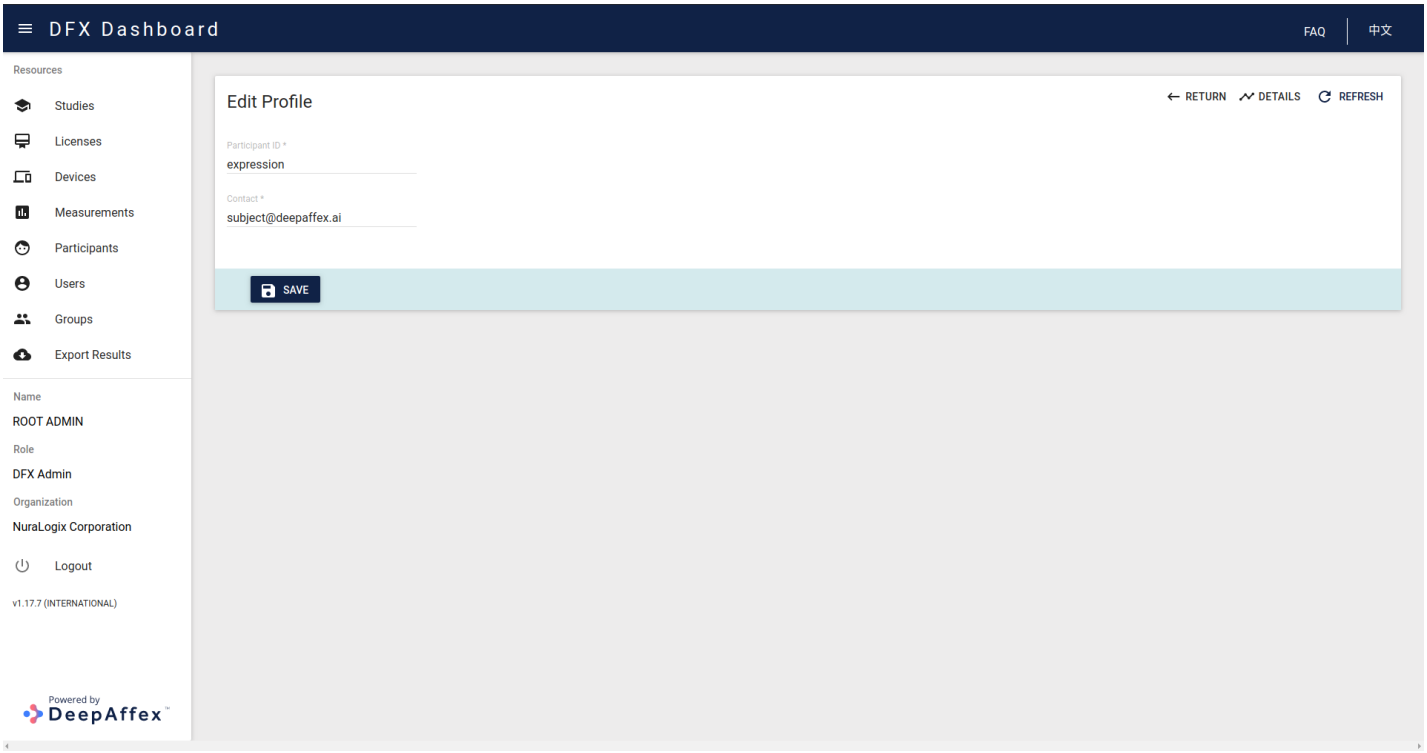
Participants

In this section you can access the attributes, measurement counts etc. for any Participants whose Measurements reside under your Organization.



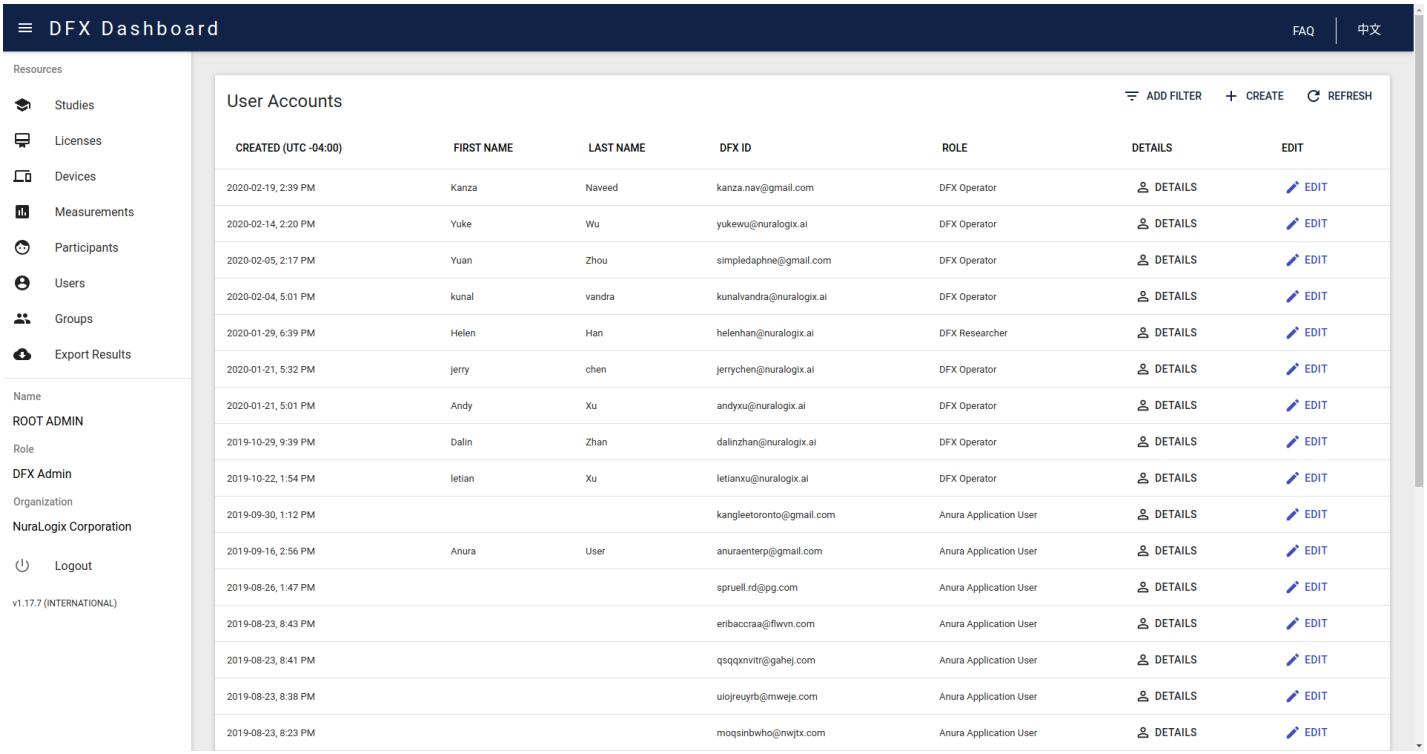
If your role is DFX Researcher or higher, you can view and edit Participants' profile details. You can also sort and filter Participants by date, DFX ID, and Participant ID.



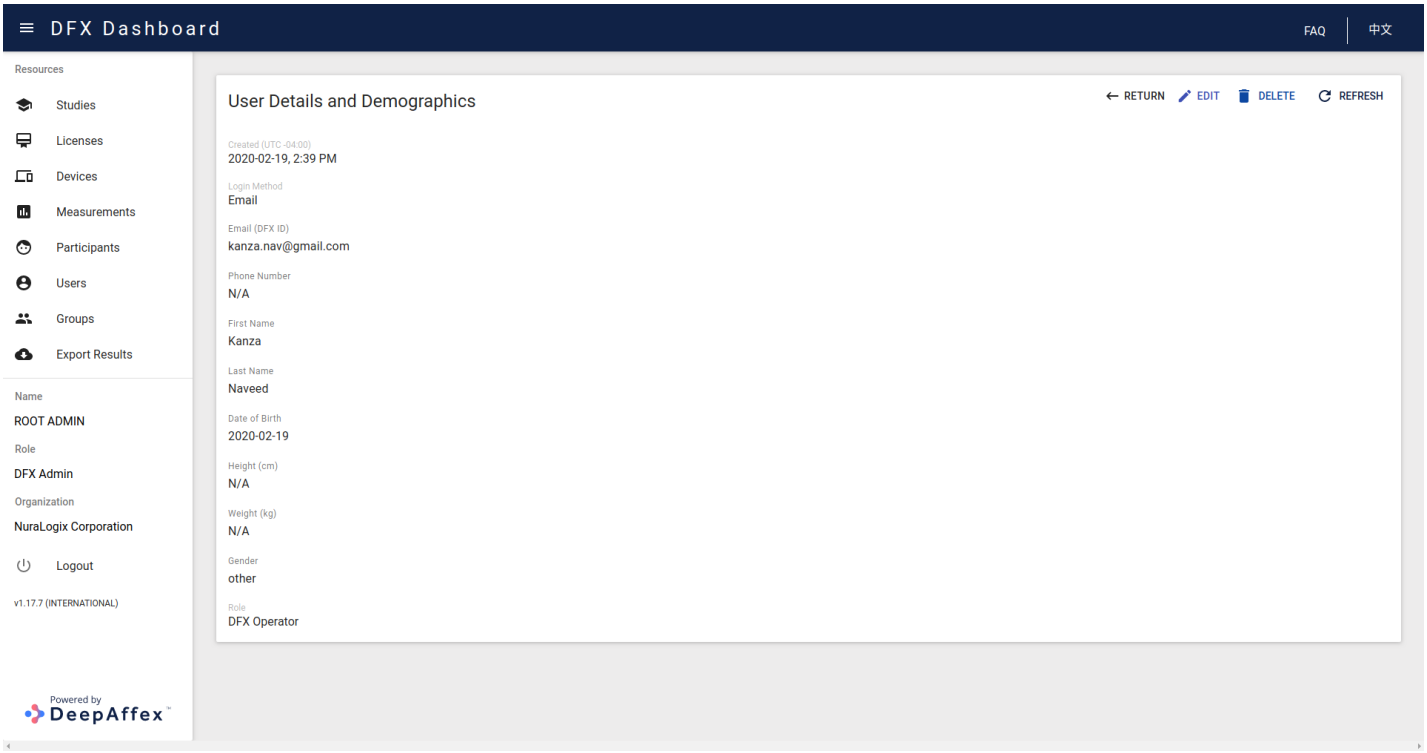

DFX Dashboard
FAQ
中文
Resources
Studies
Licenses
Devices
Measurements
Participants
Users
Groups
Export Results
Name
ROOT ADMIN
Role
DFX Admin
Organization
NuraLogix Corporation
Logout
v1.17.7 (INTERNATIONAL)
Powered by DeepAffex
Edit Profile
RETURN
DETAILS
REFRESH
Participant ID *
expression
Contact *
subject@deepaffex.ai
SAVE

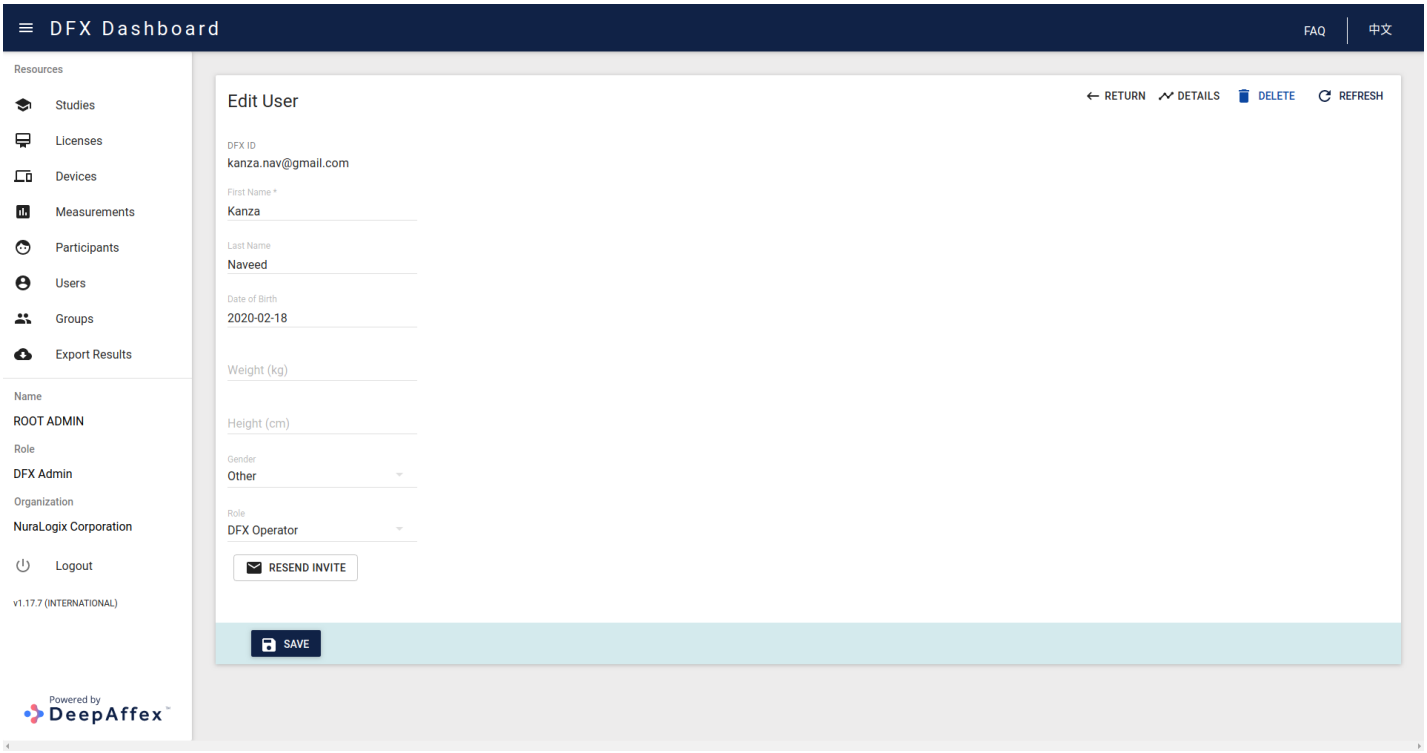
Users

In this section you can access the configuration attributes for all Users who are registered under your Organization.



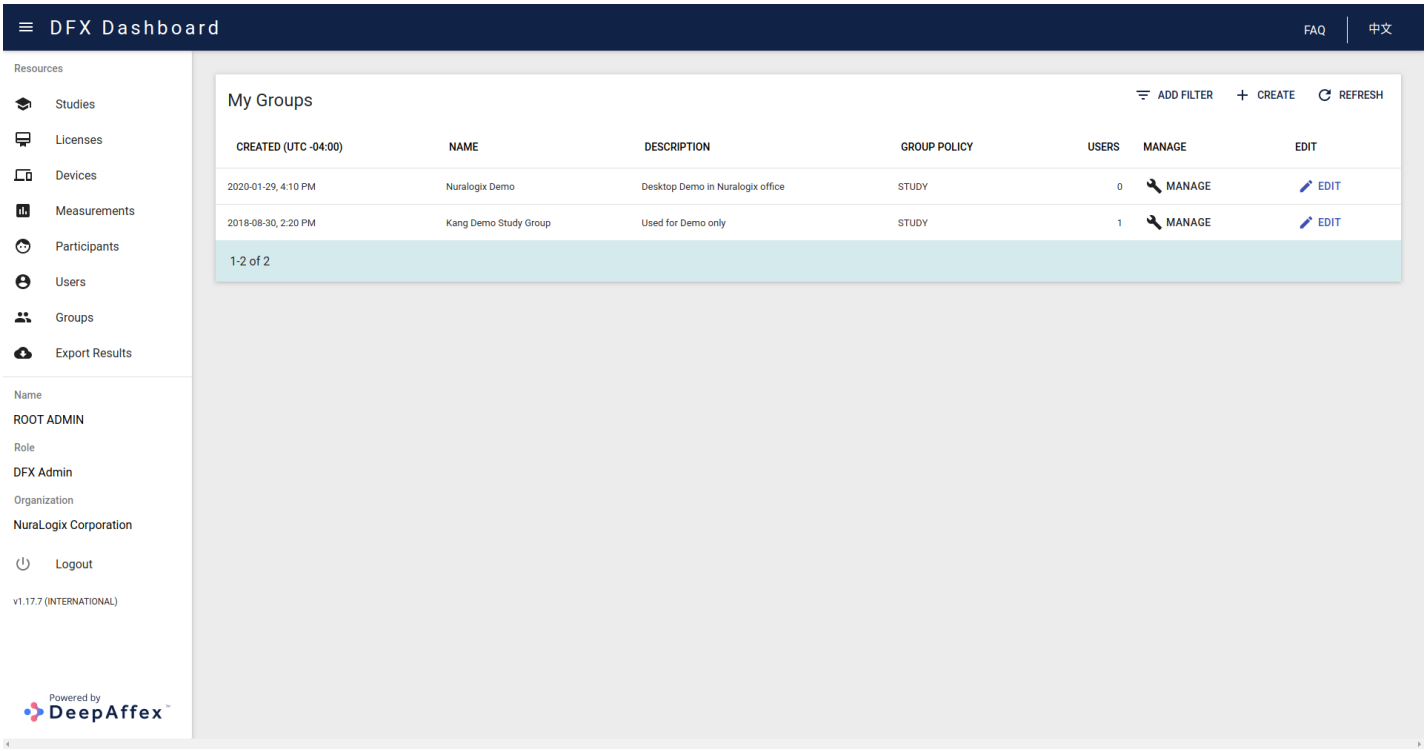
If your role is DFX Lead or higher, you can view and edit User details. You can also sort and filter Users by date, DFX ID or roles.



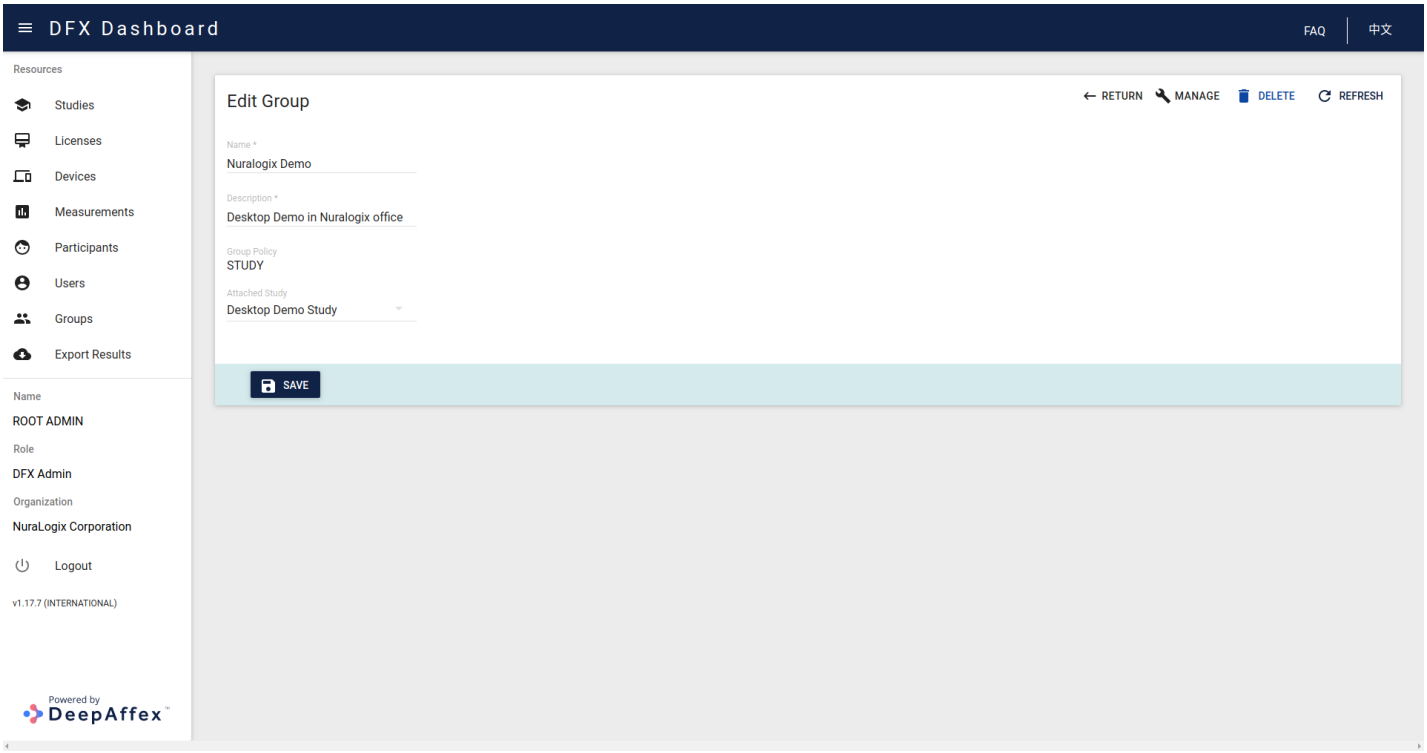

DFX Dashboard
FAQ
中文
Resources
Studies
Licenses
Devices
Measurements
Participants
Users
Groups
Export Results
Name
ROOT ADMIN
Role
DFX Admin
Organization
NuraLogix Corporation
Logout
v1.17.7 (INTERNATIONAL)
Powered by DeepAffex
Edit User
RETURN
DETAILS
DELETE
REFRESH
DFX ID
kanza.nav@gmail.com
First Name *
Kanza
Last Name
Naveed
Date of Birth
2020-02-18
Weight (kg)
Height (cm)
Gender
Other
Role
DFX Operator
RESEND INVITE
SAVE

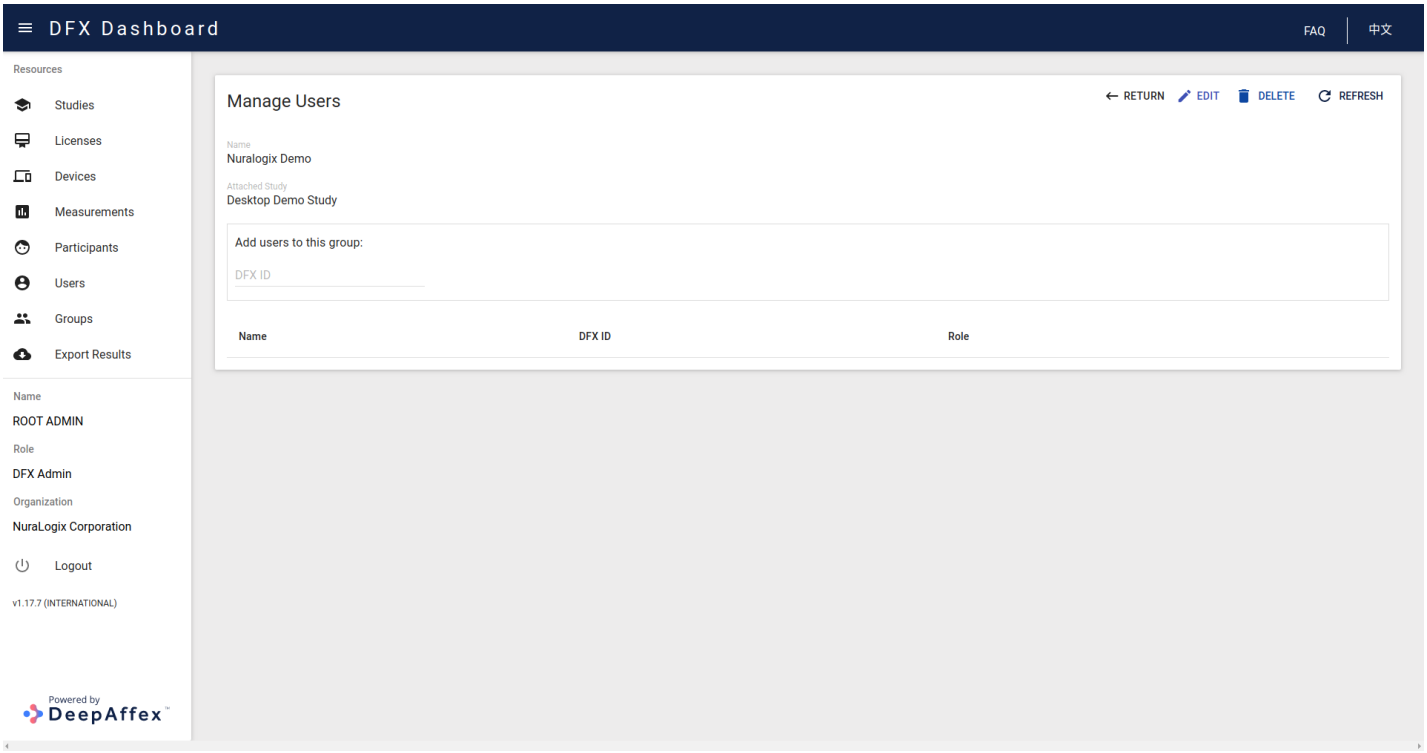
Groups

In this section you can access Group attributes and policies used to restrict the scope of access under your Organization.



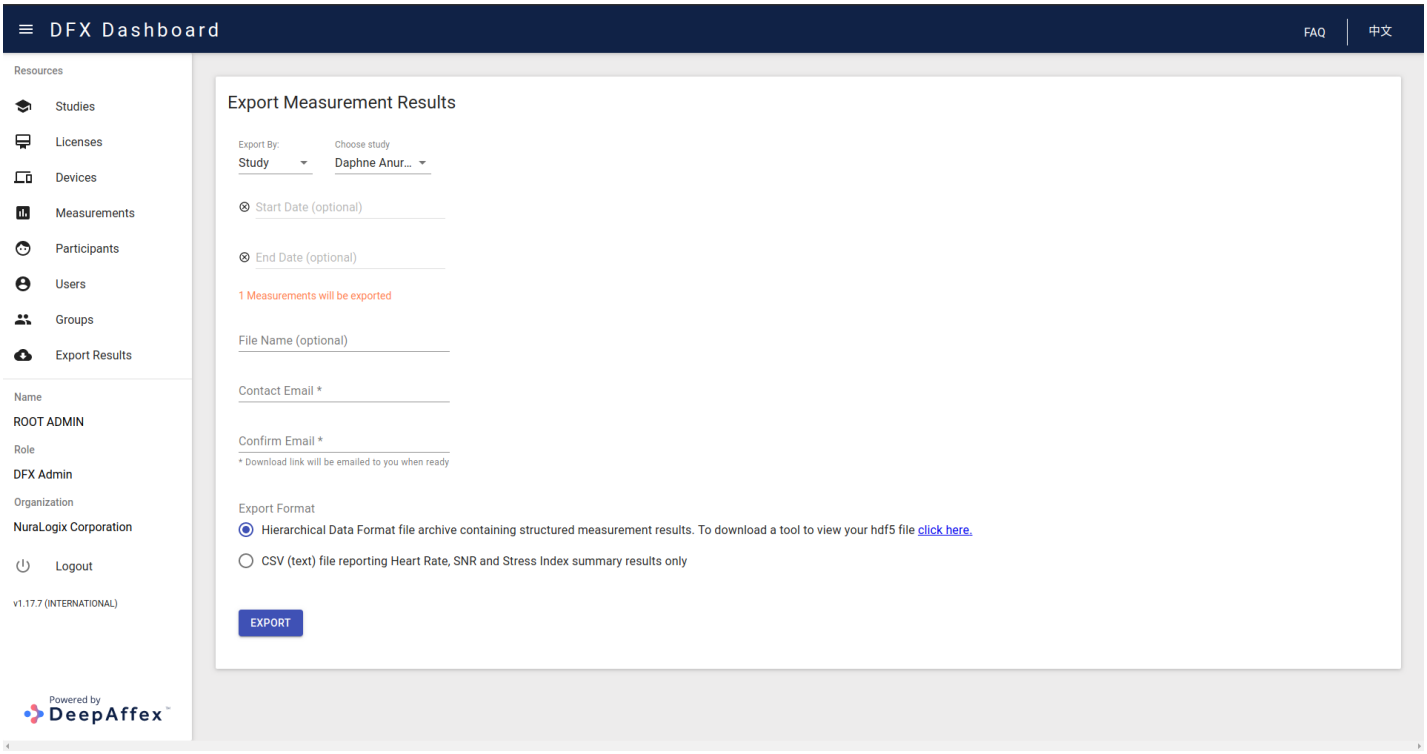
If your role is DFX Lead or higher, you can manage and edit Groups. You can also sort and filter Groups by policy types.



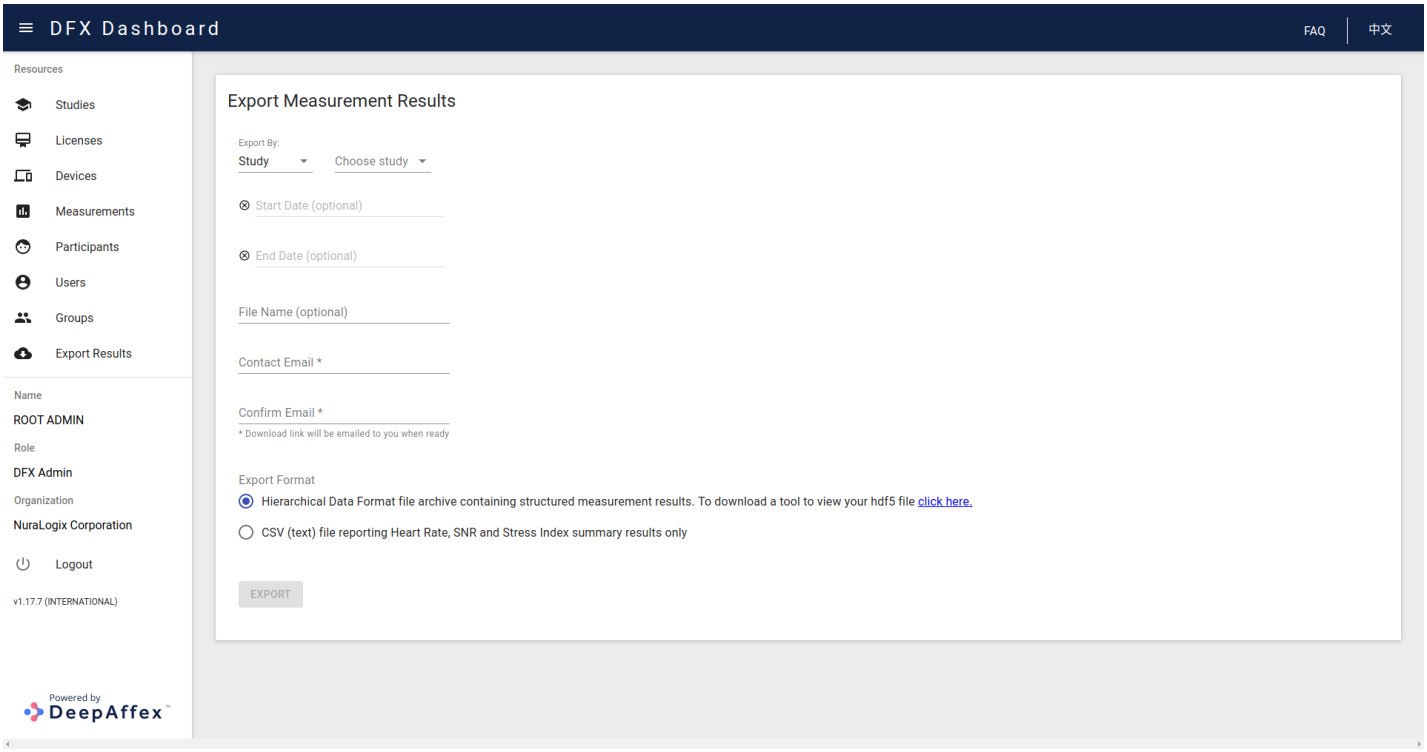

DFX Dashboard
FAQ
中文
Resources
Studies
Licenses
Devices
Measurements
Participants
Users
Groups
Export Results
Name
ROOT ADMIN
Role
DFX Admin
Organization
NuraLogix Corporation
Logout
v1.17.7 (INTERNATIONAL)
Powered by DeepAffex
Manage Users
RETURN
EDIT
DELETE
REFRESH
Name
Nuralogix Demo
Attached Study
Desktop Demo Study
Add users to this group:
DFX ID
Name
DFX ID
Role

Result Export

From this section you can export results - either in the HDF5 or CSV format. (Format choice is based on the export option of the Study, User and/or Organization.)



You can only export results, if your role is DFX Lead or higher.



DFX Points

The table below lists all available DFX Points.

ID	Name	Unit	Extraction Time (seconds)	Results Available After (seconds)
AGE	Facial Skin Age	years	1	1
BMI	Body Mass Index	kg/m²	1	1
BMI_CALC	Calculated Body Mass Index	kg/m²	0	0
BP_CVD	Cardiovascular Disease Risk	%	30	30
BP_DIASTOLIC	Diastolic Blood Pressure	mm-Hg	30	30
BP_HEART_ATTACK	Heart Attack Risk	%	30	30
BP_MAP	Mean Arterial Pressure	mm-Hg	30	30
BP_PP	Pulse Pressure	mm-Hg	30	30
BP_RPP	Cardiac Workload	dB	30	30
BP_STROKE	Stroke Risk	%	30	30
BP_SYSTOLIC	Systolic Blood Pressure	mm-Hg	30	30
BP_TAU	Vascular Capacity	seconds	30	30
BR_BPM	Breathing Rate	bpm	15	15
BR_ENVELOPE	Breathing Wave		30	30
BR_HERTZ	Breathing Rate	Hz	15	15
COVID19_SCORE	COVID-19 Score	%	0	0
GENDER	Gender	M/F	1	1

ID	Name	Unit	Extraction Time (seconds)	Results Available After (seconds)
HEALTH_SCORE	General Wellness Score		0	0
HEART_RATE	Heart Rate 140	Hz	15	15
HEIGHT	Height	cm	1	1
HR180_BPM	Heart Rate 180	bpm	15	15
HR180_HERTZ	Heart Rate 180	Hz	15	15
HR_21T	ROI 21 Heart Rate 140	bpm	15	15
HR_23T	ROI 23 Heart Rate 140	bpm	15	15
HR_BPM	Heart Rate 140	bpm	15	15
HR_ENVELOPE	Heart Rate 140 Wave		5	15
HR_HERTZ	Heart Rate 140	Hz	15	15
HRV_SDNN	Heart Rate Variability (SDNN)	ms	15	15
IHB_COUNT	Irregular Heartbeats		30	30
MAYER_ENVELOPE	Mayer Wave		30	60
MAYER_POWER	Mayer Wave Power		30	60
MENTAL_SCORE	Mental Score		0	0
MENTAL_STRESS_INDEX	NuraLogix™ Mental Stress Index		120	120

ID	Name	Unit	Extraction Time (seconds)	Results Available After (seconds)
MSI	NuraLogix™ Mental Stress Index		15	15
MSI_POWER	NuraLogix™ Mental Stress Index Power		60	120
PHYSICAL_SCORE	Physical Score		0	0
PHYSIO_SCORE	Physiological Score		0	0
RISKS_SCORE	Risks Score		0	0
RRI	Beat-to-beat Interval		30	30
SNR	Signal-to-Noise Ratio (SNR)	dB	15	15
SNR_19G	ROI 19 SNR (slow Green)	dB	15	15
SNR_19T4	ROI 19 SNR (slow TOI4)	dB	15	15
SNR_21T	ROI 21 SNR	dB	5	15
SNR_23T	ROI 23 SNR	dB	5	15
SNR_WSST	Instantaneous SNR	dB	5	15
STRESS_SCORE	NuraLogix™ Mental Stress Score		120	120
THERMAL_ENVELOPE	Thermal Wave		60	120
THERMAL_POWER	Thermal Wave Power		60	120
VITAL_SCORE	Vital Score		0	0
WEIGHT	Weight	kg	1	1

Glossary

Transdermal Optical Imaging (TOI™): A technique used to extract blood flow information from a sequence of images typically from a video stream of a person's face.

DFX Point: A biosignal of interest that can be extracted, computed or estimated from facial blood-flow data that is assigned a unique name on the DeepAffex™ Cloud. (e.g. `HR_BPM` is the DFX Point for heart rate.) A Point can be a Source, Signal, Feature, Algorithm, Model or Classifier.

- **Source**: Point calculated from a "raw" or minimally processed data stream.
- **Feature**: Point extracted by a NuraLogix™ developed feature.
- **Algorithm**: Point calculated from a NuraLogix™ developed algorithm.
- **Model**: Point estimated by a NuraLogix™ developed machine learning model.
- **Classifier**: Point produced by a NuraLogix™ developed classifier.

DFX Study Template: A group of DFX Points used to create a DFX Study. Study Templates can only be created by NuraLogix™.

DFX Study: A group of DFX Points that are measured in one DFX Measurement. A Study is based on a DFX Study Template and can be created by an Organization.

DFX Measurement: A group of results computed by the DeepAffex™ Cloud based on a single DFX Study. The process of extracting facial blood-flow from a sequence of images and sending it to the DeepAffex™ Cloud for processing is called *making a measurement*.