

DFX Extraction Library v4.15.8

NuraLogix Corporation

Contents

1 DFX Extraction library v4.15.8 1
- 1.0.1 Getting started . . . 1

2 Deprecated List 3

3 Namespace Index 5
- 3.1 Namespace List . . . 5

4 Class Index 7
- 4.1 Class List . . . 7

5 File Index 9
- 5.1 File List . . . 9

6 Namespace Documentation 11
- 6.1 dfx Namespace Reference . . . 11
  - 6.1.1 Detailed Description . . . 12
  - 6.1.2 Enumeration Type Documentation . . . 12
    - 6.1.2.1 CollectorState . . . 12
    - 6.1.2.2 ConstraintResult . . . 13

7 Class Documentation 15
- 7.1 dfx::ChunkData Class Reference . . . 15
  - 7.1.1 Detailed Description . . . 15
  - 7.1.2 Member Function Documentation . . . 15
    - 7.1.2.1 getChunkPayload() . . . 15
- 7.2 dfx::ChunkPayload Struct Reference . . . 16
  - 7.2.1 Detailed Description . . . 16
- 7.3 dfx::Collector Class Reference . . . 16
  - 7.3.1 Detailed Description . . . 18
  - 7.3.2 Member Function Documentation . . . 18
    - 7.3.2.1 cancelCollection() . . . 18
    - 7.3.2.2 checkConstraints() . . . 19
    - 7.3.2.3 createFrame() . . . 19
    - 7.3.2.4 decodeMeasurementResult() . . . 19
    - 7.3.2.5 defineRegions() . . . 20
    - 7.3.2.6 disableConstraint() . . . 20
    - 7.3.2.7 enableConstraint() . . . 21
    - 7.3.2.8 extractChannels() . . . 21
    - 7.3.2.9 forceComplete() . . . 21
    - 7.3.2.10 getAvailableConstraints() . . . 22
    - 7.3.2.11 getChunkData() . . . 22
    - 7.3.2.12 getChunkDurationSeconds() . . . 22
    - 7.3.2.13 getCollectorState() . . . 22
    - 7.3.2.14 getConstraintErrorMessage() . . . 23
    - 7.3.2.15 getConstraintsConfig() . . . 23
    - 7.3.2.16 getEnabledConstraints() . . . 23

7.3.2.17 getLastErrorMessage() . . . 24
7.3.2.18 getMode() . . . 24
7.3.2.19 getNumberChunks() . . . 24
7.3.2.20 getProperties() . . . 24
7.3.2.21 getProperty() . . . 25
7.3.2.22 getRequiredPosePointIDs() . . . 25
7.3.2.23 initializeModel() . . . 25
7.3.2.24 isChunkReady() . . . 25
7.3.2.25 numberFramesNeeded() . . . 26
7.3.2.26 prepareMeasurement() . . . 26
7.3.2.27 resetCollection() . . . 26
7.3.2.28 setChunkDurationSeconds() . . . 27
7.3.2.29 setCloudResultsFeedback() . . . 27
7.3.2.30 setConstraintsConfig() . . . 27
7.3.2.31 setFaceAttribute() [1/2] . . . 28
7.3.2.32 setFaceAttribute() [2/2] . . . 28
7.3.2.33 setNumberChunks() . . . 29
7.3.2.34 setProperty() . . . 29
7.3.2.35 setTargetFPS() . . . 29
7.3.2.36 startCollection() . . . 30
7.3.2.37 stopCollection() . . . 30
7.4 dfx::DFXFactory Class Reference . . . 30
7.4.1 Detailed Description . . . 31
7.4.2 Member Function Documentation . . . 32
7.4.2.1 addToStudy() . . . 32
7.4.2.2 addToStudyFromFile() . . . 33
7.4.2.3 createCollector() . . . 33
7.4.2.4 getLastErrorMessage() . . . 34
7.4.2.5 getMode() . . . 34
7.4.2.6 getProperty() . . . 34
7.4.2.7 getSdkID() . . . 34
7.4.2.8 getValidProperties() . . . 35
7.4.2.9 getVersion() . . . 35
7.4.2.10 initializeStudy() . . . 35
7.4.2.11 initializeStudyFromFile() . . . 36
7.4.2.12 removeProperty() . . . 36
7.4.2.13 setMode() . . . 37
7.4.2.14 setProperty() . . . 37
7.5 dfx::Error Struct Reference . . . 37
7.6 dfx::Face Struct Reference . . . 38
7.6.1 Detailed Description . . . 38
7.6.2 Member Data Documentation . . . 38
7.6.2.1 attributes . . . 38
7.6.2.2 detected . . . 39
7.6.2.3 faceRect . . . 39
7.6.2.4 id . . . 39
7.6.2.5 posePoints . . . 39
7.6.2.6 poseValid . . . 39
7.7 dfx::Frame Class Reference . . . 40
7.7.1 Detailed Description . . . 41
7.7.2 Member Function Documentation . . . 41
7.7.2.1 addFace() . . . 41
7.7.2.2 addMarker() . . . 41
7.7.2.3 getDesiredDeviceAttributes() . . . 42
7.7.2.4 getDesiredFaceAttributes() . . . 42
7.7.2.5 getDeviceAttribute() . . . 42
7.7.2.6 getFaceIdentifiers() . . . 43
7.7.2.7 getMarkers() . . . 43

©Copyright 2018-2022 NuraLogix Corporation

7.7.2.8 getOpticalQualityMetrics() . . . 43
7.7.2.9 getOpticalQualityRating() . . . 43
7.7.2.10 getQualityMetrics() . . . 44
7.7.2.11 getRegionHistogram() . . . 44
7.7.2.12 getRegionIntProperty() . . . 45
7.7.2.13 getRegionNames() . . . 45
7.7.2.14 getRegionPolygon() . . . 45
7.7.2.15 getStarRating() . . . 46
7.7.2.16 getVideoFrame() . . . 46
7.7.2.17 setDeviceAttribute() . . . 46
7.8 dfx::MeasurementData Class Reference . . . 47
7.8.1 Detailed Description . . . 47
7.8.2 Member Function Documentation . . . 47
7.8.2.1 getData() . . . 47
7.8.2.2 getDataProperty() . . . 47
7.8.2.3 getDataPropertyKeys() . . . 48
7.9 dfx::MeasurementResult Class Reference . . . 48
7.9.1 Detailed Description . . . 48
7.9.2 Member Function Documentation . . . 48
7.9.2.1 getErrorCode() . . . 48
7.9.2.2 getMeasurementData() . . . 49
7.9.2.3 getMeasurementDataKeys() . . . 49
7.9.2.4 getMeasurementProperty() . . . 49
7.9.2.5 getMeasurementPropertyKeys() . . . 50
7.9.2.6 isValid() . . . 50
7.10 dfx::PosePoint Struct Reference . . . 50
7.10.1 Detailed Description . . . 51
7.10.2 Member Data Documentation . . . 51
7.10.2.1 estimated . . . 51
7.10.2.2 point . . . 51
7.10.2.3 quality . . . 52
7.10.2.4 valid . . . 52
7.11 dfx::ThrowOnError Class Reference . . . 52
7.12 dfx::VideoFrame Struct Reference . . . 52
7.12.1 Detailed Description . . . 53
7.12.2 Member Enumeration Documentation . . . 53
7.12.2.1 ChannelOrder . . . 53
7.12.3 Member Data Documentation . . . 54
7.12.3.1 image . . . 54
7.12.3.2 number . . . 54
7.12.3.3 order . . . 54
7.12.3.4 timestamp_millisec . . . 54
7.12.3.5 timestamp_ns . . . 55

8 File Documentation **57**
8.1 ChunkData.h . . . 57
8.2 ChunkPayload.h . . . 57
8.3 Collector.h . . . 58
8.4 core.h . . . 60
8.5 DFXErrors.h . . . 60
8.6 DFXFactory.h . . . 61
8.7 Face.h . . . 62
8.8 Frame.h . . . 63
8.9 MeasurementData.h . . . 64
8.10 MeasurementResult.h . . . 64
8.11 PosePoint.h . . . 65
8.12 VideoFrame.h . . . 65

Index **67**

©Copyright 2018-2022 NuraLogix Corporation

Chapter 1

DFX Extraction library v4.15.8

The primary purpose of the Nuralogix DeepAffex™ Extraction library (DFX Extraction library) is to convert a series of face-tracked images into resultant blood-flow. This procedure is referred to as 'blood-flow extraction' and is the first stage in the Transdermal Optical Imaging (TOI) pipeline required for DeepAffex™ processing. The DFX Extraction library is used to generate measurement data chunks that are then forwarded to the DeepAffex™ Server for analysis and processing.

The DFX Extraction library 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 Ubuntu 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 C library has no external dependencies).

1.0.1 Getting started

The `dfx::DFXFactory` is the primary entry point for the C++ DFX Extraction library, and is used to create one or more `dfx::Collector`s.

The DFX Extraction library leaves it up to the client application to capture frames (from a camera or a video file) and to populate the `dfx::VideoFrame` structure. This is done so that client applications and devices can capture in any format that is most suitable to their environment.

The client application is also responsible for ensuring that `dfx::Frame`s (created from `dfx::VideoFrame`s) have accurate timestamps and `dfx::Face`s. We provide no built-in face pose estimation as it is available as a commodity with implementations like `dlib` and `Visage` or via platform-specific libraries like Apple's `Vision Framework` or Android's `FaceDetector`. Whatever the output of the face tracking engine, it will need to be transformed into `dfx::PosePoint` definitions conforming to the standard `MPEG-4 Facial Data Points` and attached to the `dfx::Face` structure.

Once sufficient `dfx::Frame`s have been added to the collector, the collector will indicate that a chunk is ready (`isChunkReady`) and the client application will need to get the `dfx::ChunkPayload` (using calls to `getChunkData` and `getChunkPayload`) and make a network call to the DeepAffex™ Server API.

The DeepAffex™ Server API leverages well formed `protobuf` structures for the wire protocol and communicates using WebSockets or REST.

©Copyright 2018-2022 NuraLogix Corporation

Chapter 2

Deprecated List

Member dfx::Collector::decodeMeasurementResult (const std::vector< uint8_t > &measurement↩ Response)=0

: Support for this method will be removed in a future. The server now returns data in JSON format avoiding the need to decode client side.

Member dfx::Collector::prepareMeasurement (const std::vector< uint8_t > &createMeasurement↩ Payload)=0

: Support for this method will be removed in a future. It has been replaced with a no-op.

Member dfx::Collector::stopCollection ()=0

replaced by cancelCollection which more accurately describes the activity since the measurement can not be resumed before being reset. Please use cancelCollection as this method will eventually be removed as it simply calls cancelCollection.

Member dfx::DFXFactory::addToStudy (const std::vector< unsigned char > &data)=0

: Support for this method will be removed in a future release. Please use the initializeStudy(const vector<unsigned char>&) to initialize a study.

Member dfx::DFXFactory::addToStudyFromFile (const std::string &pathToFile)=0

: Support for this method will be removed in a future release. Please use the initializeStudy(const vector<unsigned char>&) to initialize a study.

Member dfx::DFXFactory::initializeStudyFromFile (const std::string &pathToFile)=0

: Support for this method will be removed in a future release. Please use the initializeStudy(const vector<unsigned char>&) to initialize a study.

Member dfx::Frame::getQualityMetrics (float iso, float exposure)=0

: Support for this method will be removed in a future release. Please use the getOpticalQualityMetrics instead.

Member dfx::Frame::getStarRating (std::string &feedback)=0

: Support for this method will be removed in a future release. Please use the getOpticalQualityRating instead.

Member dfx::VideoFrame::timestamp_ns

this will be removed in favor of timestamp_millisec

©Copyright 2018-2022 NuraLogix Corporation

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

dfx 11

©Copyright 2018-2022 NuraLogix Corporation

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

dfx::ChunkData
: ChunkData is the internal memory representation of a Collection from which a ChunkPayload can be constructed to send to the DFX API 15

dfx::ChunkPayload
: Chunk that was captured by a Collection 16

dfx::Collector
: Collector extracts region channel information from video frames for server signal processing . . 16

dfx::DFXFactory
: DFXFactory is the primary entry point for the collector and is used create a collector 30

dfx::Error 37

dfx::Face
: Identifies the properties associated with faces within the video frame 38

dfx::Frame
: Frame is a wrapper which links a VideoFrame with one or more Face objects and additional state needed for regions and channel construction 40

dfx::MeasurementData 47

dfx::MeasurementResult 48

dfx::PosePoint
: Pose points are used to identify individual MPEG-4 Facial Data Points of a face 50

dfx::ThrowOnError 52

dfx::VideoFrame
: Represents the internal structure for how image frames are passed to the DFX Engine since there is little in the way of standards 52

©Copyright 2018-2022 NuraLogix Corporation

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

dfx/ChunkData.h . . . 57
dfx/ChunkPayload.h . . . 57
dfx/Collector.h . . . 58
dfx/core.h . . . 60
dfx/DFXErrors.h . . . 60
dfx/DFXFactory.h . . . 61
dfx/Face.h . . . 62
dfx/Frame.h . . . 63
dfx/MeasurementData.h . . . 64
dfx/MeasurementResult.h . . . 64
dfx/PosePoint.h . . . 65
dfx/VideoFrame.h . . . 65

©Copyright 2018-2022 NuraLogix Corporation

Chapter 6

Namespace Documentation

6.1 dfx Namespace Reference

Classes

- class ChunkData

  ChunkData is the internal memory representation of a Collection from which a ChunkPayload can be constructed to send to the DFX API.
- struct ChunkPayload

  Chunk that was captured by a Collection.
- class Collector

  Collector extracts region channel information from video frames for server signal processing.
- class DFXFactory

  DFXFactory is the primary entry point for the collector and is used create a collector.
- struct Error
- struct Face

  identifies the properties associated with faces within the video frame.
- class Frame

  Frame is a wrapper which links a VideoFrame with one or more Face objects and additional state needed for regions and channel construction.
- class MeasurementData
- class MeasurementResult
- struct PosePoint

  Pose points are used to identify individual MPEG-4 Facial Data Points of a face.
- class ThrowOnError
- struct VideoFrame

  Represents the internal structure for how image frames are passed to the DFX Engine since there is little in the way of standards.

Enumerations

- enum class ConstraintResult : char { Good = 0 , Warn = 1 , Error = 2 }

  The possible constraint return values.
- enum class CollectorState : char {
  Waiting = 0 , Collecting = 1 , ChunkReady = 2 , Completed = 3 ,
  Error = 4 }

A Collector state indicates whether a Collector is in error or ready to collect data.

- enum class **FaceAttribute** {
  SEX_ASSIGNED_AT_BIRTH = 1 , **AGE_YEARS** = 2 , **HEIGHT_CM** = 3 , **WEIGHT_KG** = 4 ,
  SMOKER = 5 , **HYPERTENSIVE** = 6 , **BLOOD_PRESSURE_MEDICATION** = 7 , **DIABETES** = 8 }
- enum class **FaceAttributeValue** {
  SEX_NOT_PROVIDED = 1 , **SEX_ASSIGNED_MALE_AT_BIRTH** = 2 , **SEX_ASSIGNED_FEMALE_AT_↩**
  BIRTH = 3 , **DIABETES_NONE** = 4 ,
  DIABETES_TYPE1 = 5 , **DIABETES_TYPE2** = 6 }

6.1.1 Detailed Description

The namespace containing all dfx related classes.

These are helpers for translating to/from the opaque dfx_error** types and follow a standard pattern to raise the exception after transversing the C-ABI layer.

See also

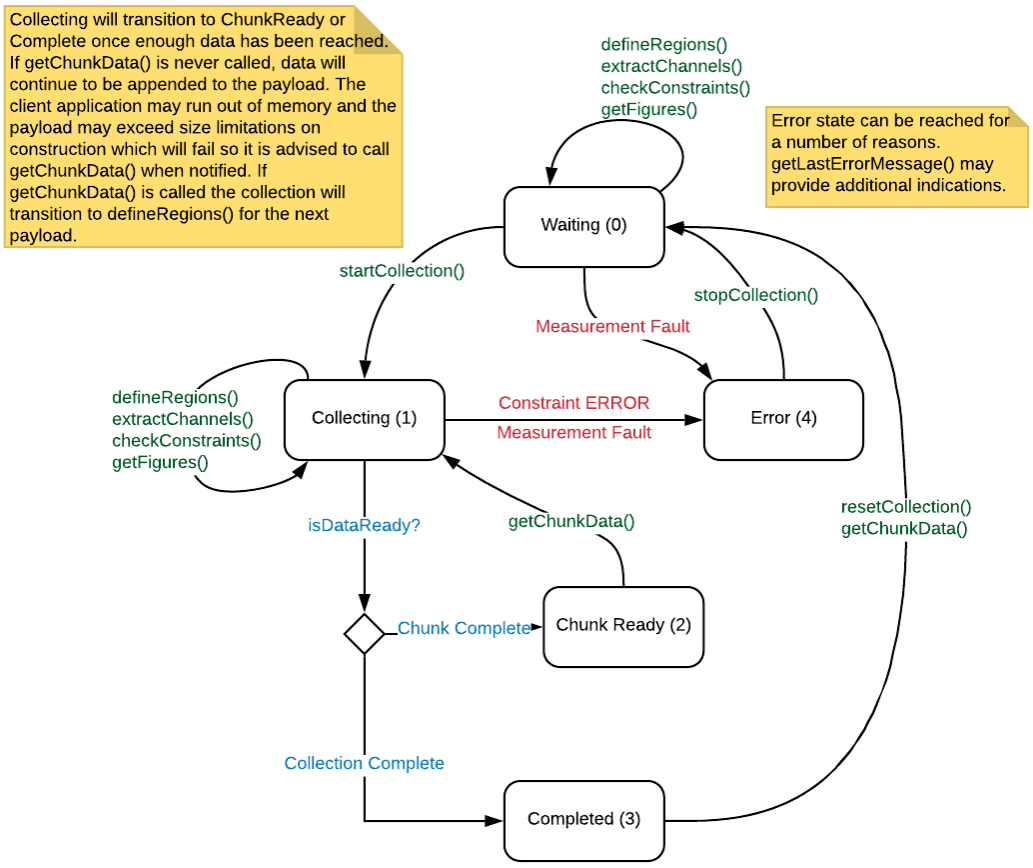
"Hourglass Interfaces for C++ APIs"

6.1.2 Enumeration Type Documentation

6.1.2.1 CollectorState

```
enum class dfx::CollectorState : char [strong]
```

A Collector state indicates whether a Collector is in error or ready to collect data.



©Copyright 2018-2022 NuraLogix Corporation

Enumerator

Waiting	Collector is WAITING to start collection
Collecting	Collector is COLLECTING frame data
ChunkReady	Collector has a CHUNK READY to send to server
Completed	Collector is COMPLETED and should be sent to server
Error	Collector in ERROR and collection aborted

6.1.2.2 ConstraintResult

```
enum class dfx::ConstraintResult : char [strong]
```

The possible constraint return values.

Enumerator

Good	Good indicates there is nothing presently detected in constraints
Warn	Warn indicates a problem that needs to be corrected
Error	Error indicates a problem that has failed the measurement

©Copyright 2018-2022 NuraLogix Corporation

©Copyright 2018-2022 NuraLogix Corporation

Chapter 7

Class Documentation

7.1 dfx::ChunkData Class Reference

ChunkData is the internal memory representation of a Collection from which a ChunkPayload can be constructed to send to the DFX API.

`#include "dfx/ChunkData.h"`

Public Member Functions

- virtual ~**ChunkData** ()

  ChunkData destructor.
- virtual ChunkPayload getChunkPayload ()=0

  The measurement payload chunk of bytes to send for the current request.

7.1.1 Detailed Description

ChunkData is the internal memory representation of a Collection from which a ChunkPayload can be constructed to send to the DFX API.

The ChunkData enables the Collector to quickly reset it's internal state between measurement chunks by handing the collected data to ChunkData.

Building the actual ChunkPayload takes some effort and time for which the Collector would otherwise be unable to start adding frames until the previous ChunkPayload had been constructed and the Collector reset.

7.1.2 Member Function Documentation

7.1.2.1 getChunkPayload()

```
virtual ChunkPayload dfx::ChunkData::getChunkPayload ()  [pure virtual]
```

The measurement payload chunk of bytes to send for the current request.

The measurement is collapsed and summarized into the payload information necessary for the server to construct the requested signal(s).

Returns

a vector of bytes to be passed by the client application to the server for the current measurement request

The documentation for this class was generated from the following file:

- dfx/ChunkData.h

7.2 dfx::ChunkPayload Struct Reference

Chunk that was captured by a Collection.

`#include "dfx/ChunkPayload.h"`

Public Attributes

- uint8_t **valid**
  if this payload was succesfully created.
- uint32_t **startFrame**
  starting frame of chunk.
- uint32_t **endFrame**
  last frame in chunk.
- uint32_t **chunkNumber**
  sequential number of the chunk starting at zero
- uint32_t **numberChunks**
  the number of chunks expected in measurement
- uint32_t **firstChunkStartTime_s**
  the start time of the first chunk in seconds
- uint32_t **startTime_s**
  start time of the chunk in seconds
- uint32_t **endTime_s**
  end time of the chunk in seconds
- float **duration_s**
  duration of the chunk in seconds
- std::vector< uint8_t > **payload**
  binary payload of chunk
- std::vector< uint8_t > **metadata**
  metadata information associated with chunk (JSON)

7.2.1 Detailed Description

Chunk that was captured by a Collection.

The documentation for this struct was generated from the following file:

- dfx/ChunkPayload.h

7.3 dfx::Collector Class Reference

Collector extracts region channel information from video frames for server signal processing.

`#include "dfx/Collector.h"`

©Copyright 2018-2022 NuraLogix Corporation

Public Member Functions

- virtual ~**Collector** ()

  Collector destructor.
- virtual std::shared_ptr< dfx::Frame > createFrame (const dfx::VideoFrame &videoFrame)=0

  Creates a Frame from a user supplied video frame.
- virtual CollectorState startCollection ()=0

  When the client is ready to start a measurement they will notify the collector to start collecting data.
- virtual bool prepareMeasurement (const std::vector< uint8_t > &createMeasurementPayload)=0

  When the client creates a new Measurement with the API, the Collector should be prepared with the API Measurement CreateResponse payload.
- virtual CollectorState getCollectorState ()=0

  The current internal state of this measurement collector.
- virtual std::string getMode () const =0

  Returns the operating mode of the Collector.
- virtual bool setTargetFPS (float targetFPS)=0

  Identifies what the anticipated FPS is suppose to be.
- virtual bool setChunkDurationSeconds (float chunkDurationSeconds)=0

  Sets the desired collected chunk duration in seconds.
- virtual float getChunkDurationSeconds ()=0

  Returns the desired collected chunk duration in seconds.
- virtual bool setNumberChunks (uint32_t numberChunks)=0

  Sets number of chunks before the measurement is complete.
- virtual uint32_t getNumberChunks ()=0

  Returns the total number of chunks the collector will process for this measurement.
- virtual bool enableConstraint (const std::string &constraintID)=0

  Enable constraint handling for the identified constraint.
- virtual bool disableConstraint (const std::string &constraintID)=0

  Disable a constraint for the identified constraint.
- virtual std::vector< std::string > getEnabledConstraints ()=0

  Returns the currently enabled constraints.
- virtual std::vector< std::string > getAvailableConstraints () const =0

  Obtain all available constraint IDs.
- virtual ConstraintResult checkConstraints (std::shared_ptr< dfx::Frame > &frame, std::vector< std::pair< std::string, dfx::ConstraintResult > > &results)=0

  Check for constraint violations.
- virtual std::string getConstraintErrorMessage (const std::string &violationID)=0

  Obtain a message which represents the violation ID.
- virtual bool setConstraintsConfig (const std::string &key, const std::string &propertyValue)=0

  Configures the constraint system properties.
- virtual std::string getConstraintsConfig (const std::string &key)=0

  Obtains a constraint system property.
- virtual std::vector< std::string > getRequiredPosePointIDs () const =0

  Return the set of pose points for the configured study.
- virtual CollectorState defineRegions (std::shared_ptr< dfx::Frame > &frame)=0

  Define the regions of interest based upon the MPEG-4 Facial Data Points.
- virtual CollectorState extractChannels (std::shared_ptr< dfx::Frame > &frame)=0

  Extracts the channels of interest from the video frame and regions of interest.
- virtual bool isChunkReady () const =0

  Identifies when a measurement has collected sufficient information to make a server request.
- virtual int32_t numberFramesNeeded () const =0

©Copyright 2018-2022 NuraLogix Corporation

The approximate number of additional video frames required before sufficient information has been collected to make the server request.

- virtual std::shared_ptr< dfx::ChunkData > getChunkData ()=0

  The measurement chunk data which has been collected up for the current chunk until this point.
- virtual std::shared_ptr< dfx::MeasurementResult > decodeMeasurementResult (const std::vector< uint8_t > &measurementResponse)=0

  Decodes the binary measurement response payload from the server into a MeasurementResult.
- virtual bool setProperty (const std::string &key, const std::string &value)=0

  Client applications wishing to associate their own metadata properties can associate key=value strings to the Collector.
- virtual std::string getProperty (const std::string &key)=0

  Returns a metadata property by key name if it exists.
- virtual void **removeProperty** (const std::string &key)=0

  Removes a metadata property by key name if it exists.
- virtual std::map< std::string, std::string > getProperties ()=0

  Obtains the Chunk metadata for this measurement request.
- virtual std::string getLastErrorMessage ()=0

  If there was an error getLastErrorMessage may contain more information about why the error occurred.
- virtual void cancelCollection ()=0

  Cancels the current collection.
- virtual void stopCollection ()=0

  Stop the current collection.
- virtual void forceComplete ()=0

  Forces the completion of a collection.
- virtual void resetCollection ()=0

  Resets the measurement state to construct a new chunk.
- virtual bool setFaceAttribute (const std::string &faceID, FaceAttribute key, FaceAttributeValue value)=0

  Add user data information to a payload using FaceAttribute key and values.
- virtual bool setFaceAttribute (const std::string &faceID, FaceAttribute key, double value)=0

  Add user data information to a payload using FaceAttribute key and numeric/boolean value.
- virtual bool setCloudResultsFeedback (const std::string &jsonCloudResults)=0

  Provide the cloud results to the Collector to potentially improve the result.
- virtual bool initializeModel (int8_t modelType, const std::vector< unsigned char > &data)=0

  Configures a model within the Collector.

7.3.1 Detailed Description

Collector extracts region channel information from video frames for server signal processing.

A Collector involves significant processing on video frames using located faces to identify regions of interest, extract and summarize the data for server signal processing.

7.3.2 Member Function Documentation

7.3.2.1 cancelCollection()

```
virtual void dfx::Collector::cancelCollection ()  [pure virtual]
```

Cancels the current collection.

When a collection violates as constraint or otherwise needs to stop this method can be called placing the collector back into a WAIT state until the client calls startCollection again.

©Copyright 2018-2022 NuraLogix Corporation

7.3.2.2 checkConstraints()

```
virtual ConstraintResult dfx::Collector::checkConstraints (
            std::shared_ptr< dfx::Frame > & frame,
            std::vector< std::pair< std::string, dfx::ConstraintResult > > & results)  [pure
virtual]
```

Check for constraint violations.

When some constraints are enabled and a constraint is detected as being violated that constraint will be added to the violations list.

Parameters

frame	the video frame currently being operated on
results	is set of constraints and associated state

Returns

result of the constraint check

7.3.2.3 createFrame()

```
virtual std::shared_ptr< dfx::Frame > dfx::Collector::createFrame (
            const dfx::VideoFrame & videoFrame)  [pure virtual]
```

Creates a Frame from a user supplied video frame.

Parameters

videoFrame	is the video Frame to wrap.

Returns

a reference to a Frame.

7.3.2.4 decodeMeasurementResult()

```
virtual std::shared_ptr< dfx::MeasurementResult > dfx::Collector::decodeMeasurementResult (
            const std::vector< uint8_t > & measurementResponse)  [pure virtual]
```

Decodes the binary measurement response payload from the server into a MeasurementResult.

Deprecated : Support for this method will be removed in a future. The server now returns data in JSON format avoiding the need to decode client side.

©Copyright 2018-2022 NuraLogix Corporation

Parameters

measurementResponse	from the server to decode

Returns

the MeasurementResult object

7.3.2.5 defineRegions()

```
virtual CollectorState dfx::Collector::defineRegions (
            std::shared_ptr< dfx::Frame > & frame)  [pure virtual]
```

Define the regions of interest based upon the MPEG-4 Facial Data Points.

A Collector leverages many regions of interest which are areas of a detected face to locate the required signals. This defineRegions call will analyze and build the candidate region set which the channels will be constructed from.

Parameters

frame	the video frame currently being operated on

Returns

the Collector state after define regions

7.3.2.6 disableConstraint()

```
virtual bool dfx::Collector::disableConstraint (
            const std::string & constraintID)  [pure virtual]
```

Disable a constraint for the identified constraint.

If the constraint was currently enabled, it will be removed from the set of currently active constraints. If it was not active or does not exist, the request is ignored.

Parameters

constraint	to disable

Returns

true if the constraintID was changeable, false otherwise.

©Copyright 2018-2022 NuraLogix Corporation

7.3.2.7 enableConstraint()

```
virtual bool dfx::Collector::enableConstraint (
            const std::string & constraintID)  [pure virtual]
```

Enable constraint handling for the identified constraint.

Some constraint validation can be done on the frames provided by the client to ensure they meet sufficient criteria to satisfy server data quality.

This may include standard checks for things like face presence and frame rate along with movement, lighting, etc.

Parameters

constraint	to enable

Returns

true if the constraintID was changeable, false otherwise.

7.3.2.8 extractChannels()

```
virtual CollectorState dfx::Collector::extractChannels (
            std::shared_ptr< dfx::Frame > & frame)  [pure virtual]
```

Extracts the channels of interest from the video frame and regions of interest.

A Collector will collapse the candidate regions of interest down to a set of channels of interest which the server will further refine into a set of signals.

Parameters

frame	the video frame currently being operated on

Returns

the Collector state after extract channels

7.3.2.9 forceComplete()

```
virtual void dfx::Collector::forceComplete ()  [pure virtual]
```

Forces the completion of a collection.

Collections will trigger a ChunkAvailable when enough data has been collected. When processing a video file, the final chunk may be insufficient in length and never trigger the ChunkAvailable state. forceComplete() will force the collector into a measurement Complete state and create a final payload with any remaining state.

©Copyright 2018-2022 NuraLogix Corporation

7.3.2.10 getAvailableConstraints()

```
virtual std::vector< std::string > dfx::Collector::getAvailableConstraints () const  [pure
virtual]
```

Obtain all available constraint IDs.

The set of available known constraint IDs which can be enabled by calling setEnableConstraints.

Returns

the vector of available constraint IDs.

7.3.2.11 getChunkData()

```
virtual std::shared_ptr< dfx::ChunkData > dfx::Collector::getChunkData ()  [pure virtual]
```

The measurement chunk data which has been collected up for the current chunk until this point.

The collected measurement chunk data is offloaded to the ChunkData and resets the internal collector state to continue processing more frames in the next chunk.

Returns

ChunkData collected to this point or null if the data is was not valid to create a chunk. ie. chunk duration elapsed but the number of valid frames was insufficient to form a useful payload.

7.3.2.12 getChunkDurationSeconds()

```
virtual float dfx::Collector::getChunkDurationSeconds ()  [pure virtual]
```

Returns the desired collected chunk duration in seconds.

Returns

the chunk durations in seconds.

7.3.2.13 getCollectorState()

```
virtual CollectorState dfx::Collector::getCollectorState ()  [pure virtual]
```

The current internal state of this measurement collector.

The internal state of this measurement collector. This is the same state returned by defineRegions or extract↩ Channels.

Once the measurement is in error state, it will remain in ERROR until reset which will place it back in WAITING.

Returns

the current internal state.

©Copyright 2018-2022 NuraLogix Corporation

7.3.2.14 getConstraintErrorMessage()

```
virtual std::string dfx::Collector::getConstraintErrorMessage (
            const std::string & violationID)  [pure virtual]
```

Obtain a message which represents the violation ID.

When presenting on a GUI, the returned constaint message is a more meaningful representation of the constraint violation ID.

Parameters

violationID	the constraint ID to obtain the error message for

Returns

error message of the corresponding violationID.

7.3.2.15 getConstraintsConfig()

```
virtual std::string dfx::Collector::getConstraintsConfig (
            const std::string & key)  [pure virtual]
```

Obtains a constraint system property.

Returns the constraint value associated with the key.

Parameters

key	the configuration key to work on, currently "json"

Returns

the value of the configuration key or an empty string if key did not exist.

7.3.2.16 getEnabledConstraints()

```
virtual std::vector< std::string > dfx::Collector::getEnabledConstraints ()  [pure virtual]
```

Returns the currently enabled constraints.

Returns

a list of the currently enabled constraint IDs.

© Copyright 2018-2022 NuraLogix Corporation

7.3.2.17 getLastErrorMessage()

```
virtual std::string dfx::Collector::getLastErrorMessage ()  [pure virtual]
```

If there was an error getLastErrorMessage may contain more information about why the error occurred.

Returns

the last known error

7.3.2.18 getMode()

```
virtual std::string dfx::Collector::getMode () const  [pure virtual]
```

Returns the operating mode of the Collector.

The mode is configured in the Factory prior to Collector creation.

Returns

the operating mode for current collector.

7.3.2.19 getNumberChunks()

```
virtual uint32_t dfx::Collector::getNumberChunks ()  [pure virtual]
```

Returns the total number of chunks the collector will process for this measurement.

Returns

the total number of chunks.

7.3.2.20 getProperties()

```
virtual std::map< std::string, std::string > dfx::Collector::getProperties ()  [pure virtual]
```

Obtains the Chunk metadata for this measurement request.

The returned chunk metadata is a superset of the metadata provided by the client application. Additional measurement details are augmented into the chunk metadata.

Returns

a map containing the metadata key/value strings

©Copyright 2018-2022 NuraLogix Corporation

7.3.2.21 getProperty()

```
virtual std::string dfx::Collector::getProperty (
            const std::string & key)  [pure virtual]
```

Returns a metadata property by key name if it exists.

The returned string will be the value of the current chunk metadata property.

Returns

a string containing the property value, or an empty string if the key does not exist.

7.3.2.22 getRequiredPosePointIDs()

```
virtual std::vector< std::string > dfx::Collector::getRequiredPosePointIDs () const  [pure
virtual]
```

Return the set of pose points for the configured study.

Once a study has been initialized this method will return the list of pose points required for each found face.

Returns

the set of MPEG-4 Facial Data Points required for each face.

7.3.2.23 initializeModel()

```
virtual bool dfx::Collector::initializeModel (
            int8_t modelType,
            const std::vector< unsigned char > & data)  [pure virtual]
```

Configures a model within the Collector.

Parameters

modelType	A numeric identifier associated with the model implementation
data	Configuration data associated with the model

Returns

true if the model was configured, false if it was not (invalid model, models not supported)

7.3.2.24 isChunkReady()

```
virtual bool dfx::Collector::isChunkReady () const  [pure virtual]
```

Identifies when a measurement has collected sufficient information to make a server request.

Returns

true if the chunk is ready to be sent, false otherwise

©Copyright 2018-2022 NuraLogix Corporation

7.3.2.25 numberFramesNeeded()

```
virtual int32_t dfx::Collector::numberFramesNeeded () const  [pure virtual]
```

The approximate number of additional video frames required before sufficient information has been collected to make the server request.

The number frames needed is a rough guide for the client application to know how many more data points will be required for a request to be sent. It is not a guarantee as variance will occur for a number of reasons: variable frame rate, participant movement, along with environment and camera impacts. When the measurement has collected enough data this will be zero and isChunkReady will return true. The number of frames is not necessarily monotomically decreasing.

Returns

the anticipated number of frames remaining or zero if sufficient information has been collected. If there is an error or inability to calculate, -1 is returned.

7.3.2.26 prepareMeasurement()

```
virtual bool dfx::Collector::prepareMeasurement (
            const std::vector< uint8_t > & createMeasurementPayload)  [pure virtual]
```

When the client creates a new Measurement with the API, the Collector should be prepared with the API Measurement CreateResponse payload.

Deprecated : Support for this method will be removed in a future. It has been replaced with a no-op.

Parameters

createMeasurementPayload	the payload from DFX server

Returns

true if collector was happy, false otherwise

7.3.2.27 resetCollection()

```
virtual void dfx::Collector::resetCollection ()  [pure virtual]
```

Resets the measurement state to construct a new chunk.

The Collector maintains internal buffers and state when constructing a chunk which needs to be reset when a new chunk collection starts. A client application can discard a collection and all accumulated state by calling reset().

©Copyright 2018-2022 NuraLogix Corporation

7.3.2.28 setChunkDurationSeconds()

```
virtual bool dfx::Collector::setChunkDurationSeconds (
            float chunkDurationSeconds)  [pure virtual]
```

Sets the desired collected chunk duration in seconds.

Identifies to the collector what the desired chunk duration should be in terms of seconds. This will typically ensure that a chunk is ready for collecting whenever the duration has been reached.

Parameters

chunkDurationSeconds	the chunk duration in seconds.

Returns

true if the chunk durations was successfully changed, false otherwise.

7.3.2.29 setCloudResultsFeedback()

```
virtual bool dfx::Collector::setCloudResultsFeedback (
            const std::string & jsonCloudResults)  [pure virtual]
```

Provide the cloud results to the Collector to potentially improve the result.

If the results obtained from the cloud are available to the Collector, it can make additional optimizations on future payloads and provide more informed feedback.

Parameters

jsonCloudResults	the JSON results obtained from the Cloud API

Returns

true if result data was valid, false if result data was not valid.

7.3.2.30 setConstraintsConfig()

```
virtual bool dfx::Collector::setConstraintsConfig (
            const std::string & key,
            const std::string & propertyValue)  [pure virtual]
```

Configures the constraint system properties.

The constraint system configuration accepts a key "json" and the corresponding JSON payload to configure the system.

©Copyright 2018-2022 NuraLogix Corporation

Parameters

key	the configuration key to work on, currently "json"
property	the value to set this property to

Returns

true if the property was successfully set, false on error

7.3.2.31 setFaceAttribute() [1/2]

```
virtual bool dfx::Collector::setFaceAttribute (
            const std::string & faceID,
            FaceAttribute key,
            double value)  [pure virtual]
```

Add user data information to a payload using FaceAttribute key and numeric/boolean value.

Parameters

faceID	the face identifier to use, typically "1"
key	the attribute key identifying the value being provided
value	the attribute value associated with key

Returns

true if value was applied, false if invalid value or unable to apply.

7.3.2.32 setFaceAttribute() [2/2]

```
virtual bool dfx::Collector::setFaceAttribute (
            const std::string & faceID,
            FaceAttribute key,
            FaceAttributeValue value)  [pure virtual]
```

Add user data information to a payload using FaceAttribute key and values.

Parameters

faceID	the face identifier to use, typically "1"
key	the attribute key identifying the value being provided
value	the attribute value associated with key

Returns

true if value was applied, false if invalid value or unable to apply.

©Copyright 2018-2022 NuraLogix Corporation

7.3.2.33 setNumberChunks()

```
virtual bool dfx::Collector::setNumberChunks (
            uint32_t numberChunks)  [pure virtual]
```

Sets number of chunks before the measurement is complete.

Parameters

numberChunks	the number of chunks.

Returns

true if the number of chunks was successfully changed, false otherwise.

7.3.2.34 setProperty()

```
virtual bool dfx::Collector::setProperty (
            const std::string & key,
            const std::string & value)  [pure virtual]
```

Client applications wishing to associate their own metadata properties can associate key=value strings to the Collector.

If there is metadata that a client applications wishes to tag against a Collector request, ie. barcode or participant name they should invoke setProperty() with the appropriate value to ensure that it is properly encoded into the Chunk Payload.

Parameters

key	a key identifier for the metadata. If the key already exists it will be replaced
value	the value to associate with the key

Returns

true if property was set, false otherwise

7.3.2.35 setTargetFPS()

```
virtual bool dfx::Collector::setTargetFPS (
            float targetFPS)  [pure virtual]
```

Identifies what the anticipated FPS is suppose to be.

The anticipated target FPS defaults to 30.0 FPS, but if a video frame rate of 60.0 FPS or higher is required it can be customized. While the actual FPS is established based on the individual VideoFrame timestamps, the anticipated FPS assists with model calibration.

Post-processing a pre-recorded video file will generally have a more consistent FPS and yield less noisy results then a live extraction which needs to share CPU processing power for video frame handling with video frame processing.

©Copyright 2018-2022 NuraLogix Corporation

If the target or actual FPS falls outside model tolerances either too high or too low the collection will be rejected.

Parameters

targetFPS	the anticipated target FPS

Returns

true if the targetFPS was successfully changed, false otherwise.

7.3.2.36 startCollection()

```
virtual CollectorState dfx::Collector::startCollection ()  [pure virtual]
```

When the client is ready to start a measurement they will notify the collector to start collecting data.

Prior to starting, a measurement should be in the WAITING state. A call to resetCollector will always ensure the measurement is in WAITING state.

While in a WAITING state, regions can be obtained and constraints analyzed for violations prior to a collection starting.

Returns

the current internal state, which should be COLLECTING unless an ERROR occurred.

7.3.2.37 stopCollection()

```
virtual void dfx::Collector::stopCollection ()  [pure virtual]
```

Stop the current collection.

When a collection violates as constraint or otherwise needs to stop this method can be called placing the collector back into a WAIT state until the client calls startCollection again.

Deprecated replaced by cancelCollection which more accurately describes the activity since the measurement can not be resumed before being reset. Please use cancelCollection as this method will eventually be removed as it simply calls cancelCollection.

The documentation for this class was generated from the following file:

- dfx/Collector.h

7.4 dfx::DFXFactory Class Reference

DFXFactory is the primary entry point for the collector and is used create a collector.

```
#include "dfx/DFXFactory.h"
```

©Copyright 2018-2022 NuraLogix Corporation

Public Member Functions

- virtual ~**DFXFactory** ()

  DFXFactory destructor.
- virtual std::string getVersion () const =0

  The version ID of this DFX Extraction library.
- virtual std::string getSdkID () const =0

  The ID of this DFX Extraction library.
- virtual bool setMode (const std::string &mode)=0

  Set the operating mode which will be used when creating a new collector.
- virtual std::string getMode () const =0

  Returns the current operating mode when creating collectors.
- virtual std::string getLastErrorMessage ()=0

  If there was an error getLastErrorMessage may contain more information about why the erorr occurred.
- virtual void setProperty (const std::string &key, const std::string &value)=0

  Specifies a configurable property key/value pair to configure the DFX Extraction library.
- virtual std::string getProperty (const std::string &key)=0

  Returns a property value for a key.
- virtual void removeProperty (const std::string &key)=0

  Removes a property from the factory if it exists.
- virtual std::vector< std::string > getValidProperties () const =0

  Obtain a list of valid property keys which can be configured.
- virtual bool initializeStudy (const std::vector< unsigned char > &data)=0

  Initializes the factory to create collectors for a study with a memory based data definition.
- virtual bool addToStudy (const std::vector< unsigned char > &data)=0

  Adds to the study definition when a definition is comprised of multiple parts with an additional memory based definition.
- virtual bool initializeStudyFromFile (const std::string &pathToFile)=0

  Initializes the factory to create collectors for a study with a file based data definition.
- virtual bool addToStudyFromFile (const std::string &pathToFile)=0

  Adds to the study definition when a definition is comprised of multiple parts with an additional file based definition.
- virtual std::shared_ptr< dfx::Collector > createCollector ()=0

  Creates a Collector to process video frames and build a request payload.

7.4.1 Detailed Description

DFXFactory is the primary entry point for the collector and is used create a collector.

Applications are able to analyze video stream and summarize relevent portions of the stream which can be sent using the DFX API to the DFX Server for additional signal processing and model prediction.

Typical usage of the DFX API and DFX Extraction library involves only a few application level calls.

While the DFX Extraction library does not perform server calls, it does construct the payloads which a client platform will send to the server. The DFX Extraction library also expects to be initialized with the server response to properly initialize the engine.

```
// Obtain a study reference from the DFX API
auto studyResponse = Application::getAPIStudy();
std::vector<unsigned int> study =

dfx::DFXFactory factory;    // Create the DFX factory instance
factory.initializeStudy(study);

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

©Copyright 2018-2022 NuraLogix Corporation

```
collector->setTargetFPS(30.0f);

collector->prepareMeasurement(measurementResponsePayload);

// Enable all possible client contraints to minimize server side rejections
collector->setEnableConstraints(collector->getAvailableConstraintIDs());

collector->startCollection();

int frameNumber = 0;
cv::Mat image;
cv::VideoCapture video("theVideoFile.mp4");
while( video.read(image) ) {

   dfx::VideoFrame videoFrame;
   videoFrame.image = image;
   videoFrame.channelOrder = dfx::VideoFormat::BGRA;
   videoFrame.timestamp = std::chrono::high_resolution_clock::now();
   videoFrame.number = frameNumber++;

   auto frame = collector->createFrame(videoFrame);

   // Application provides face tracking and builds face
   // and adds it to frame.
   auto face = Application::getDFXFace(videoFrame);
   frame->addFace(face);

   // Add markers, acceleration, rotation etc. if needed
   frame->addMarker("smile");

   // Define the collector regions of interest for this frame.
   collector->defineRegions(frame);

   std::vector<std::string> violations;
   auto result = collector->checkConstraints(frame, violations);
   if( result != ConstraintResult::Error ) {

     // Extract the channels for this frame
     collector->extractChannels(frame);

     // Different studies have different requirements on number of
     // frames to produce a chunk.
     if ( collector->isChunkReady() ) {
        auto chunkData = collector->getChunkData();

        auto chunkPayload = chunkData->getChunkPayload();

        std::vector<uint8_t> metadata = chunkPayload.metadata;
        std::vector<uint8_t> payload  = chunkPayload.payload;

        // This is where the Application uses the DFX API to make
        // the network call to the DFX Server.
        auto payload = Application::getAPIcollectorPayload(metadata, bytes);
        auto prediction = Application::getAPIPrediction(payload);
     }
   } else {
     collector->resetCollection();
   }
}
```

7.4.2 Member Function Documentation

7.4.2.1 addToStudy()

```
virtual bool dfx::DFXFactory::addToStudy (
             const std::vector< unsigned char > & data)  [pure virtual]
```

Adds to the study definition when a definition is comprised of multiple parts with an additional memory based definition.

Deprecated : Support for this method will be removed in a future release. Please use the initializeStudy(const vector<unsigned char>&) to initialize a study.

©Copyright 2018-2022 NuraLogix Corporation

If this is the first call, it will initialize the study with the definition otherwise it will be added to the existing data definitions.

Parameters

data	to add to the study definition

Returns

true if the the data passes some basic sanity, false otherwise.

See also

DFXNetworkAPI

7.4.2.2 addToStudyFromFile()

```
virtual bool dfx::DFXFactory::addToStudyFromFile (
            const std::string & pathToFile)  [pure virtual]
```

Adds to the study definition when a definition is comprised of multiple parts with an additional file based definition.

Deprecated : Support for this method will be removed in a future release. Please use the initializeStudy(const vector<unsigned char>&) to initialize a study.

If this is the first call, it will initialize the study with the definition otherwise it will be added to the existing data definitions.

Parameters

pathToFile	to add to the study definition

Returns

true if the the data passes some basic sanity, false otherwise.

See also

DFXNetworkAPI

7.4.2.3 createCollector()

```
virtual std::shared_ptr< dfx::Collector > dfx::DFXFactory::createCollector ()  [pure virtual]
```

Creates a Collector to process video frames and build a request payload.

Returns

a reference to a Collector or nullptr on failure.

©Copyright 2018-2022 NuraLogix Corporation

7.4.2.4 getLastErrorMessage()

```
virtual std::string dfx::DFXFactory::getLastErrorMessage ()  [pure virtual]
```

If there was an error getLastErrorMessage may contain more information about why the erorr occurred.

Returns

the last known error

7.4.2.5 getMode()

```
virtual std::string dfx::DFXFactory::getMode () const  [pure virtual]
```

Returns the current operating mode when creating collectors.

Returns

the currently configured operating mode

7.4.2.6 getProperty()

```
virtual std::string dfx::DFXFactory::getProperty (
            const std::string & key)  [pure virtual]
```

Returns a property value for a key.

Parameters

key	to obtain the property value for.

Returns

the property if it exists or an empty string.

7.4.2.7 getSdkID()

```
virtual std::string dfx::DFXFactory::getSdkID () const  [pure virtual]
```

The ID of this DFX Extraction library.

When requesting a study file from the DFX server it requires the ID which is returned by this method.

Since

4.3.13

Returns

the ID of this DFX Extraction library.

©Copyright 2018-2022 NuraLogix Corporation

7.4.2.8 getValidProperties()

```
virtual std::vector< std::string > dfx::DFXFactory::getValidProperties () const  [pure virtual]
```

Obtain a list of valid property keys which can be configured.

Returns

list of configurable property keys.

See also

setProperty

7.4.2.9 getVersion()

```
virtual std::string dfx::DFXFactory::getVersion () const  [pure virtual]
```

The version ID of this DFX Extraction library.

Returns

the version ID of this DFX Extraction library.

7.4.2.10 initializeStudy()

```
virtual bool dfx::DFXFactory::initializeStudy (
            const std::vector< unsigned char > & data)  [pure virtual]
```

Initializes the factory to create collectors for a study with a memory based data definition.

An instance of a DFXFactory will only create collectors for the most recently configured study. Previously initialized study definitions are replaced with the new data if true is returned.

Parameters

data	from the server which identifies the study characteristics this DFXFactory will use.

Returns

true if the the data passes some basic sanity, false otherwise.

See also

DFXNetworkAPI

©Copyright 2018-2022 NuraLogix Corporation

7.4.2.11 initializeStudyFromFile()

```
virtual bool dfx::DFXFactory::initializeStudyFromFile (
            const std::string & pathToFile)  [pure virtual]
```

Initializes the factory to create collectors for a study with a file based data definition.

Deprecated : Support for this method will be removed in a future release. Please use the initializeStudy(const vector<unsigned char>&) to initialize a study.

An instance of a DFXFactory will only create collectors for the most recently configured study. Previously initialized study definitions are replaced with the new data if true is returned.

Parameters

pathToFile	from the server which identifies the study characteristics this DFXFactory will use.

Returns

true if the the data passes some basic sanity, false otherwise.

See also

DFXNetworkAPI

7.4.2.12 removeProperty()

```
virtual void dfx::DFXFactory::removeProperty (
            const std::string & key)  [pure virtual]
```

Removes a property from the factory if it exists.

Parameters

key	of property to remove if it exists, ignored if it does not exist.

See also

getValidProperties, setProperties

©Copyright 2018-2022 NuraLogix Corporation

7.4.2.13 setMode()

```
virtual bool dfx::DFXFactory::setMode (
            const std::string & mode)  [pure virtual]
```

Set the operating mode which will be used when creating a new collector.

Identifies the collector as being a "streaming" or a "discrete" collector. By default the collector is considered "discrete".

Parameters

mode	the string representing the operating mode.

Returns

true if mode was successfully changed, false otherwise.

7.4.2.14 setProperty()

```
virtual void dfx::DFXFactory::setProperty (
            const std::string & key,
            const std::string & value)  [pure virtual]
```

Specifies a configurable property key/value pair to configure the DFX Extraction library.

Parameters

key	of property to configure.
value	of property to configure.

See also

getValidProperties

The documentation for this class was generated from the following file:

- dfx/DFXFactory.h

7.5 dfx::Error Struct Reference

Public Attributes

- dfx_error * **opaque**

The documentation for this struct was generated from the following file:

- dfx/DFXErrors.h

©Copyright 2018-2022 NuraLogix Corporation

7.6 dfx::Face Struct Reference

identifies the properties associated with faces within the video frame.

`#include "dfx/Face.h"`

Public Attributes

- std::string id

  the identity of this face.
- cv::Rect faceRect

  the bounding rectangle of the face.
- bool poseValid

  if the pose points are valid for this face.
- bool detected

  if the face was detected or assumed based on prior knowledge.
- std::map< std::string, dfx::PosePoint > posePoints

  the PosePoint information for this face.
- std::map< std::string, double > attributes

  are used to augment the face with additional data.

7.6.1 Detailed Description

identifies the properties associated with faces within the video frame.

All faces must be represented by a unique id in order to correlate faces across video frames, if you only have one face it can be something as simple as "1".

7.6.2 Member Data Documentation

7.6.2.1 attributes

`std::map<std::string, double> dfx::Face::attributes`

are used to augment the face with additional data.

While it is not required, applications can provide additional face attributes for studies which know how to interpret the labels.

An example of this might be "age" or "weight" of the individual depicted by the face is known, the attributes can include that metric data.

Some libraries provide "pitch", "roll", and "yaw" for a face.

All studies have a known set of attribute labels which they expect or optionally can accept and ignore all others.

©Copyright 2018-2022 NuraLogix Corporation

7.6.2.2 detected

```
bool dfx::Face::detected
```

if the face was detected or assumed based on prior knowledge.

It is used to help DFX know how much it should trust this face data when making predictions.

7.6.2.3 faceRect

```
cv::Rect dfx::Face::faceRect
```

the bounding rectangle of the face.

Identifies the crude bounding box of a face. Face detection algorithms will typically identify the face rect and then perform pose estimation within the faceRect to obtain more detailed pose points for a face.

7.6.2.4 id

```
std::string dfx::Face::id
```

the identity of this face.

All faces must be uniquely identified within the DFX Engine. If a facial recognition engine is being used, the name of the individual can be used otherwise a GUID or counter may be used.

7.6.2.5 posePoints

```
std::map<std::string, dfx::PosePoint> dfx::Face::posePoints
```

the PosePoint information for this face.

PosePoint information is required in order for more accurate predictions which build regions of interest based on the defined PosePoints.

7.6.2.6 poseValid

```
bool dfx::Face::poseValid
```

if the pose points are valid for this face.

In general, passing invalid poses will cause the Measurement constraints to be violated and the Measurement will need to be reset in order to continue.

The documentation for this struct was generated from the following file:

- dfx/Face.h

©Copyright 2018-2022 NuraLogix Corporation

7.7 dfx::Frame Class Reference

Frame is a wrapper which links a VideoFrame with one or more Face objects and additional state needed for regions and channel construction.

`#include "dfx/Frame.h"`

Public Member Functions

- virtual ~**Frame** ()

  Frame destructor.
- virtual VideoFrame getVideoFrame () const =0

  Convenience method to retrieve the VideoFrame associated with this Frame instance.
- virtual bool addFace (std::shared_ptr< dfx::Face > face)=0

  Add a face and associated data to the frame.
- virtual std::vector< float > getQualityMetrics (float iso, float exposure)=0

  Calculates the quality metrics for the current frame.
- virtual std::vector< float > getOpticalQualityMetrics ()=0

  Calculates the optical quality metrics for the current frame.
- virtual int getStarRating (std::string &feedback)=0

  Provide the 5-star rating value for the current frame, available after a call to getQualityMetrics.
- virtual int getOpticalQualityRating (std::string &feedback)=0

  Provide the 5-star rating value for the current frame, available after a call to getOpticalQualityMetrics.
- virtual bool addMarker (const std::string &label)=0

  Adds a marker/label to a video frame.
- virtual std::vector< std::string > getMarkers () const =0

  Returns the marker/labels in a video frame.
- virtual std::vector< std::string > getDesiredFaceAttributes () const =0

  Returns a list of desired face attributes for this frame.
- virtual std::vector< std::string > getDesiredDeviceAttributes () const =0

  Returns a list of desired device attributes for this frame.
- virtual double getDeviceAttribute (const std::string &key) const =0

  Returns the value of a device attribute if it exists or zero.
- virtual void setDeviceAttribute (const std::string &key, double value)=0

  add a device attribute
- virtual std::vector< std::string > getFaceIdentifiers () const =0

  obtain the known face identifiers within this frame.
- virtual std::vector< std::string > getRegionNames (const std::string &faceID) const =0

  obtains the names of regions which are defined for the face within this frame.
- virtual std::vector< cv::Point > getRegionPolygon (const std::string &faceID, const std::string &regionName) const =0

  obtain the polygon for the region of interest.
- virtual int32_t getRegionIntProperty (const std::string &faceID, const std::string &regionName, const std::↩ string &property) const =0

  Regions can have numeric properties attached to them.
- virtual cv::Mat getRegionHistogram (const std::string &faceID, const std::string &regionName) const =0

  obtain the histogram for a region of interest.

©Copyright 2018-2022 NuraLogix Corporation

7.7.1 Detailed Description

Frame is a wrapper which links a VideoFrame with one or more Face objects and additional state needed for regions and channel construction.

In addition to the video image, a Frame will have Faces and addition state. It also enables client applications to augment the Frame with markers and device data like Acceleration and Rotation data if available.

7.7.2 Member Function Documentation

7.7.2.1 addFace()

```
virtual bool dfx::Frame::addFace (
            std::shared_ptr< dfx::Face > face)  [pure virtual]
```

Add a face and associated data to the frame.

Multiple faces can be added if an image contains multiple faces.

Parameters

face	data to add to this Frame.

Returns

true if face successfully added, false if the limit on the number of faces has been reached.

7.7.2.2 addMarker()

```
virtual bool dfx::Frame::addMarker (
            const std::string & label)  [pure virtual]
```

Adds a marker/label to a video frame.

If multiple events occur within a frame, multiple markers can be addedd and annotated against a frame.

Parameters

label	the string marker identifier to add to the frame.

Returns

true if marker succesfully added, false if the limit on the number of markers has been reached.

©Copyright 2018-2022 NuraLogix Corporation

7.7.2.3 getDesiredDeviceAttributes()

```
virtual std::vector< std::string > dfx::Frame::getDesiredDeviceAttributes () const  [pure
virtual]
```

Returns a list of desired device attributes for this frame.

This enables the DFX engine to request additional properties like "acceleration", "rotation", etc. which it would like to have but which might not always be available depending upon the use case.

These attributes should be added with frame setDeviceAttribute() if they are available for the current frame.

Returns

the list of desired attribute names

See also

setDeviceAttribute

7.7.2.4 getDesiredFaceAttributes()

```
virtual std::vector< std::string > dfx::Frame::getDesiredFaceAttributes () const  [pure virtual]
```

Returns a list of desired face attributes for this frame.

This enables the DFX engine to request additional properties like "age", "gender", "weight" etc. which it would like to have but which might not always be available depending upon the use case.

These attributes should be added to the Face attribute map if they are available for the current frame.

Returns

the list of desired attribute names

7.7.2.5 getDeviceAttribute()

```
virtual double dfx::Frame::getDeviceAttribute (
            const std::string & key) const  [pure virtual]
```

Returns the value of a device attribute if it exists or zero.

Parameters

key	the attribute name.

Returns

value the value of the attribute or zero.

©Copyright 2018-2022 NuraLogix Corporation

7.7.2.6 getFaceIdentifiers()

```
virtual std::vector< std::string > dfx::Frame::getFaceIdentifiers () const  [pure virtual]
```

obtain the known face identifiers within this frame.

Every face is known by an identifier which uniquely identifies the face.

Returns

vector of face IDs currently known to this frame.

7.7.2.7 getMarkers()

```
virtual std::vector< std::string > dfx::Frame::getMarkers () const  [pure virtual]
```

Returns the marker/labels in a video frame.

Returns

returns a list of markers in this frame.

7.7.2.8 getOpticalQualityMetrics()

```
virtual std::vector< float > dfx::Frame::getOpticalQualityMetrics ()  [pure virtual]
```

Calculates the optical quality metrics for the current frame.

Returns

the quality metrics

7.7.2.9 getOpticalQualityRating()

```
virtual int dfx::Frame::getOpticalQualityRating (
            std::string & feedback)  [pure virtual]
```

Provide the 5-star rating value for the current frame, available after a call to getOpticalQualityMetrics.

Parameters

the	feedback string

Returns

the 5-star rating

See also

getQualityMetrics

©Copyright 2018-2022 NuraLogix Corporation

7.7.2.10 getQualityMetrics()

```
virtual std::vector< float > dfx::Frame::getQualityMetrics (
            float iso,
            float exposure)  [pure virtual]
```

Calculates the quality metrics for the current frame.

Deprecated : Support for this method will be removed in a future release. Please use the getOpticalQualityMetrics instead.

The provided ISO and exposure settings along with the current frame image data are used to construct a set of quality metrics which are used by the exposure algorithm to determine if the ISO needs to be increased or decreased to improve the image quality.

Parameters

iso	the current iso setting
exposure	the current exposure setting

Returns

the quality metrics

7.7.2.11 getRegionHistogram()

```
virtual cv::Mat dfx::Frame::getRegionHistogram (
            const std::string & faceID,
            const std::string & regionName) const  [pure virtual]
```

obtain the histogram for a region of interest.

Parameters

faceID	identifier for the face of interest.
regionName	which histogram is desired.

Returns

histogram for the region if it is known, empty cv::Mat if not known.

©Copyright 2018-2022 NuraLogix Corporation

7.7.2.12 getRegionIntProperty()

```
virtual int32_t dfx::Frame::getRegionIntProperty (
            const std::string & faceID,
            const std::string & regionName,
            const std::string & property) const  [pure virtual]
```

Regions can have numeric properties attached to them.

Parameters

faceID	identifier for the face of interest.
regionName	to inspect the property of.
property	to be returned

Returns

the numeric value of the property.

7.7.2.13 getRegionNames()

```
virtual std::vector< std::string > dfx::Frame::getRegionNames (
            const std::string & faceID) const  [pure virtual]
```

obtains the names of regions which are defined for the face within this frame.

Parameters

faceID	identifier for the face of interest.

Returns

vector of region names which can be retrieved for the face.

7.7.2.14 getRegionPolygon()

```
virtual std::vector< cv::Point > dfx::Frame::getRegionPolygon (
            const std::string & faceID,
            const std::string & regionName) const  [pure virtual]
```

obtain the polygon for the region of interest.

Parameters

faceID	identifier for the face of interest.
regionName	which points are needed for drawing.

Returns

vector of points forming a closed polygon shape for region of interest.

© Copyright 2018-2022 NuraLogix Corporation

7.7.2.15 getStarRating()

```
virtual int dfx::Frame::getStarRating (
            std::string & feedback)  [pure virtual]
```

Provide the 5-star rating value for the current frame, available after a call to getQualityMetrics.

Deprecated : Support for this method will be removed in a future release. Please use the getOpticalQualityRating instead.

Parameters

the	feedback string

Returns

the 5-star rating

See also

getQualityMetrics

7.7.2.16 getVideoFrame()

```
virtual VideoFrame dfx::Frame::getVideoFrame () const  [pure virtual]
```

Convenience method to retrieve the VideoFrame associated with this Frame instance.

return the associated VideoFrame

7.7.2.17 setDeviceAttribute()

```
virtual void dfx::Frame::setDeviceAttribute (
            const std::string & key,
            double value)  [pure virtual]
```

add a device attribute

Device attributes are things like "acceleration" or "rotation".

When these attributes involve multiple values, like X and Y co-ordinates they should be expanded as "acceleration:x" and "acceleration:y".

Parameters

key	the attribute name.
value	the value of the attribute.

The documentation for this class was generated from the following file:

- dfx/Frame.h

©Copyright 2018-2022 NuraLogix Corporation

7.8 dfx::MeasurementData Class Reference

`#include "dfx/MeasurementData.h"`

Public Member Functions

- virtual std::vector< std::string > getDataPropertyKeys ()=0

  Returns a list of measurement data property keys that can be used with getDataProperty().
- virtual std::string getDataProperty (const std::string &key)=0

  Returns a measurement data property by key.
- virtual std::vector< double > getData ()=0

  Returns the numerical data associated with this MeasurmentData.

7.8.1 Detailed Description

A MeasurementData contains MeasurementResult data for a specific signal.

7.8.2 Member Function Documentation

7.8.2.1 getData()

```
virtual std::vector< double > dfx::MeasurementData::getData ()  [pure virtual]
```

Returns the numerical data associated with this MeasurmentData.

The vector can contain one or more values based upon the study definition and what the server is able to provide.

For instance, if the MeasurementData is for "HEART_RATE" this may be one value which would be a summary for the entire measurement chunk or it could be an estimated value per-frame of the measurement chunk.

Returns

a vector of bytes containing the data of the MeasurementData.

7.8.2.2 getDataProperty()

```
virtual std::string dfx::MeasurementData::getDataProperty (
            const std::string & key)  [pure virtual]
```

Returns a measurement data property by key.

Measurement data may include additional properties which are available based upon what is returned by getDataPropertyKeys().

This may include for instance "ID" which could be "HEART_RATE" if the signal this MeasurementData represents is associated with "HEART_RATE".

Returns

the value of the property, or an empty string if key doesn't exist.

©Copyright 2018-2022 NuraLogix Corporation

7.8.2.3 getDataPropertyKeys()

```
virtual std::vector< std::string > dfx::MeasurementData::getDataPropertyKeys ()  [pure virtual]
```

Returns a list of measurement data property keys that can be used with getDataProperty().

Returns

list of measurement data property keys

The documentation for this class was generated from the following file:

- dfx/MeasurementData.h

7.9 dfx::MeasurementResult Class Reference

```
#include "dfx/MeasurementResult.h"
```

Public Member Functions

- virtual uint8_t isValid ()=0
- virtual std::vector< std::string > getMeasurementPropertyKeys ()=0
- virtual std::string getMeasurementProperty (const std::string &key)=0
- virtual std::vector< std::string > getMeasurementDataKeys ()=0
- virtual std::shared_ptr< MeasurementData > getMeasurementData (const std::string &key)=0
- virtual std::string getErrorCode ()=0

7.9.1 Detailed Description

A MeasurementResult is the decoded MeasurementResult from the server and contains properties along with individual MeasurementData for the various signals available in the study.

7.9.2 Member Function Documentation

7.9.2.1 getErrorCode()

```
virtual std::string dfx::MeasurementResult::getErrorCode ()  [pure virtual]
```

Returns the error code associated with the measurement.

If the MeasurementResult isValid(), this will be "OK" otherwise it will contain another code which identifies an error like "INTERNAL_ERROR".

Returns

the error code associated with the measurement result.

©Copyright 2018-2022 NuraLogix Corporation

7.9.2.2 getMeasurementData()

```
virtual std::shared_ptr< MeasurementData > dfx::MeasurementResult::getMeasurementData (
            const std::string & key)  [pure virtual]
```

Obtains the MeasurementData for a known key.

If the key does not exist, a nullptr will be returned.

Parameters

key	to obtain the MeasurementData for

Returns

the MeasurementData

7.9.2.3 getMeasurementDataKeys()

```
virtual std::vector< std::string > dfx::MeasurementResult::getMeasurementDataKeys ()  [pure
virtual]
```

Returns a list of available data keys contained in this measurement result.

The actual list is a function of the study definition and what the server was able to generate based upon the data provided. This is a convenience method to assist in inspecting the measurment data result, but if you know the keys of interest those can be asked for directly using getMeasurementData().

Possible keys might include "HEART_RATE", "SNR", etc.

Returns

a list of keys present in this measurement result

7.9.2.4 getMeasurementProperty()

```
virtual std::string dfx::MeasurementResult::getMeasurementProperty (
            const std::string & key)  [pure virtual]
```

Obtain individual properties associated with this MeasurementResult.

These properties are useful when communicating with the server about the Measurement. Specifically, it will include "MeasurementID", "MeasurementDataID", "MeasurementResultID" and possibly others.

Parameters

key	the measurement property to look up

Returns

the valid of the measurement property or an empty string

©Copyright 2018-2022 NuraLogix Corporation

7.9.2.5 getMeasurementPropertyKeys()

```
virtual std::vector< std::string > dfx::MeasurementResult::getMeasurementPropertyKeys ()
[pure virtual]
```

Returns a list of measurement property keys that can be used with getMeasurementProperty().

Returns

list of measurement property keys

7.9.2.6 isValid()

```
virtual uint8_t dfx::MeasurementResult::isValid ()  [pure virtual]
```

Returns true if this measurement result is valid, otherwise returns false.

Returns

true (non-zero) if measurement result is valid, otherwise false (zero)

The documentation for this class was generated from the following file:

- dfx/MeasurementResult.h

7.10 dfx::PosePoint Struct Reference

Pose points are used to identify individual MPEG-4 Facial Data Points of a face.

```
#include "dfx/PosePoint.h"
```

Public Attributes

- cv::Point3f point

  the X,Y,Z point location information.
- bool valid

  if this point is valid, false otherwise.
- bool estimated

  if this point is "well known", or estimated based on other points.
- float quality

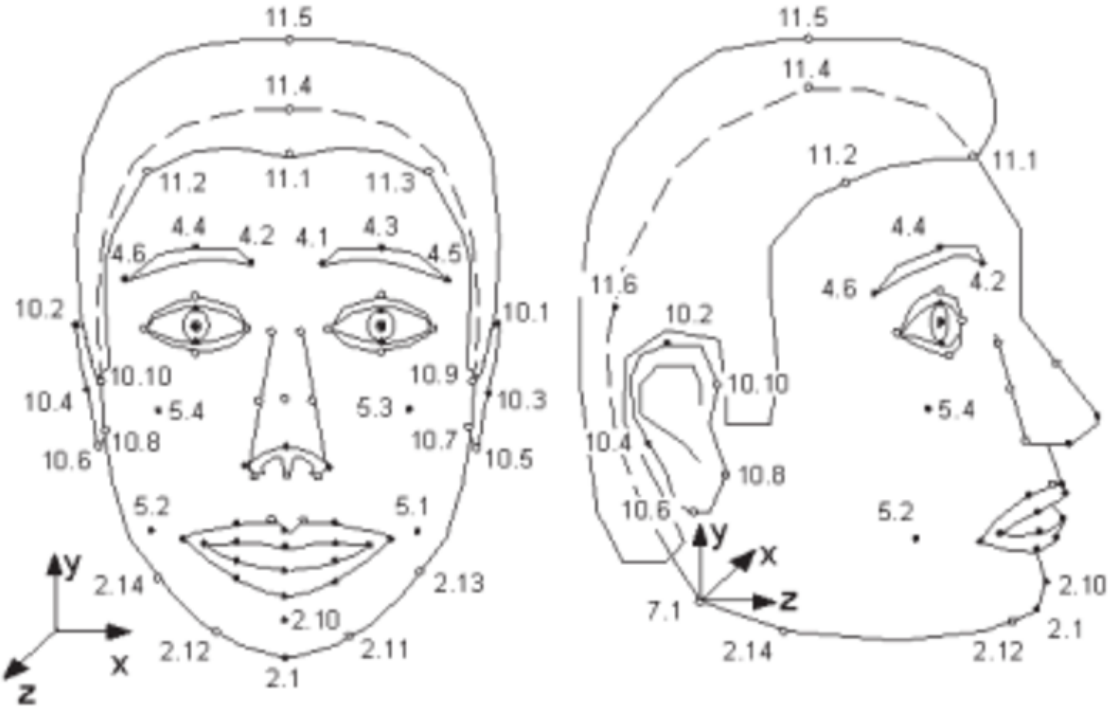
  the probability quality probability of this point between zero and one.

©Copyright 2018-2022 NuraLogix Corporation

7.10.1 Detailed Description

Pose points are used to identify individual MPEG-4 Facial Data Points of a face.

DFX utilizes a subset of the 84 feature points described in the MPEG-4 Facial Data Points when constructing regions of interest.



See also

MPEG-4 Facial Data Points

7.10.2 Member Data Documentation

7.10.2.1 estimated

```
bool dfx::PosePoint::estimated
```

if this point is "well known", or estimated based on other points.

When the MPEG-4 feature points are based on machine learning techniques, this value would be false as they are assumed to be more strongly known. On the otherhand, if this point was extrapolated or estimated based on other points that were machine learned, this flag should be set to true.

7.10.2.2 point

```
cv::Point3f dfx::PosePoint::point
```

the X,Y,Z point location information.

Presently, the Z is ignored for all study processing as the image is fundamentally 2-D and we are able to construct accurate regions from just the X and Y.

©Copyright 2018-2022 NuraLogix Corporation

7.10.2.3 quality

```
float dfx::PosePoint::quality
```

the probability quality probability of this point between zero and one.

Providing this information can help the region definitions chose it's anchor points more accurately.

7.10.2.4 valid

```
bool dfx::PosePoint::valid
```

if this point is valid, false otherwise.

Because a face maybe partially obscured, not all points may be valid within a specified face and this flag is used to indicate those points which are not valid.

The documentation for this struct was generated from the following file:

- dfx/PosePoint.h

7.11 dfx::ThrowOnError Class Reference

Public Member Functions

- **operator dfx_error** ∗∗ ()

The documentation for this class was generated from the following file:

- dfx/DFXErrors.h

7.12 dfx::VideoFrame Struct Reference

Represents the internal structure for how image frames are passed to the DFX Engine since there is little in the way of standards.

```
#include "dfx/VideoFrame.h"
```

Public Types

- enum class ChannelOrder {
  BGR = 1 , RGB = 2 , BGRA = 3 , RGBA = 4 ,
  Infrared = 5 , Infrared888 = 6 , BGR_Infrared = 7 , RGB_Infrared = 8 ,
  Gray = 9 }

©Copyright 2018-2022 NuraLogix Corporation

Public Attributes

- cv::Mat image

  the image color channels for the frame.
- ChannelOrder order

  the image channel ordering.
- uint64_t timestamp_ns

  the timestamp of this video frame.
- uint32_t number

  the video frame number within the video sequence.
- double timestamp_millisec {std::numeric_limits<double>::quiet_NaN()}

  the timestamp of the video frame in milliseconds

7.12.1 Detailed Description

Represents the internal structure for how image frames are passed to the DFX Engine since there is little in the way of standards.

DFX requires having all three color channels and an accurate timestamp for the video frame along with knowing it's position within the stream.

7.12.2 Member Enumeration Documentation

7.12.2.1 ChannelOrder

```
enum class dfx::VideoFrame::ChannelOrder  [strong]
```

is used to identify the channel order of a VideoFrame.

Typically, OpenCV uses BGRA but different client platforms have different internal representations which can be identified through the ChannelOrder format so the library knows where to find the channels of interest when processing regions of interest.

Enumerator

BGR	channels are provided in blue, green, red order
RGB	channels are provided in red, green, blue order
BGRA	channels are provided in blue, green, red, alpha order
RGBA	channels are provided in red, green, blue, alpha order
Infrared	channels are provided as 8-bit infrared
Infrared888	channels are provided as 3-channels of 8-bit infrared
BGR_Infrared	channels are provided in blue, green, red, infrared order
RGB_Infrared	channels are provided in red, green, blue, infrared order
Gray	channels are provided as gray scale

©Copyright 2018-2022 NuraLogix Corporation

7.12.3 Member Data Documentation

7.12.3.1 image

```
cv::Mat dfx::VideoFrame::image
```

the image color channels for the frame.

OpenCV cv::Mat structure is used as a convenience when specifying the image data.

7.12.3.2 number

```
uint32_t dfx::VideoFrame::number
```

the video frame number within the video sequence.

Frame numbers are expected to be sequentially provided for a measurement and the number of sequential frames must be sufficient for the study measurement requirements to be fulfilled. Since multiple measurements can be performed on a video sequence, the frame number helps identify positioning within a video and if a frame is lost.

7.12.3.3 order

```
ChannelOrder dfx::VideoFrame::order
```

the image channel ordering.

The order is used to help identify the internal format of the cv::Mat structure since it internally doesn't provide a way to identified the color channels.

7.12.3.4 timestamp_millisec

```
double dfx::VideoFrame::timestamp_millisec {std::numeric_limits<double>::quiet_NaN()}
```

the timestamp of the video frame in milliseconds

When this value is provided, not NaN, it will be used and timestamp_ns will be ignored. The addition of this while keeping timestamp_ns is to allow for a brief transition period.

The value has type double to allow for fractions of a millisecond and is analogous to how the web handles DOMHighResTimeStamp.

Migrating code from timestamp_ns to timestamp_millisec can be done by converting from:

```
frame.timestamp_ns = getTimestamp()
```

to: frame.timstamp_millisec = getTimestamp()/static_cast<double>(1e6);

Accurate DFX predictions require a consistent and accurate timestamp for the video frames. This is easily achievable when processing offline videos, but will require some effort for processing real-time camera feeds to ensure the frames maintain a consistent inter-frame interval.

Since

4.7

See also

https://developer.mozilla.org/en-US/docs/Web/API/DOMHighResTimeStamp

©Copyright 2018-2022 NuraLogix Corporation

7.12.3.5 timestamp_ns

```
uint64_t dfx::VideoFrame::timestamp_ns
```

the timestamp of this video frame.

Accurate DFX predictions require a consistent and accurate timestamp for the video frames. This is easily achievable when processing offline videos, but will require some effort for processing real-time camera feeds to ensure the frames maintain a consistent inter-frame interval.

Deprecated this will be removed in favor of timestamp_millisec

The documentation for this struct was generated from the following file:

- dfx/VideoFrame.h

©Copyright 2018-2022 NuraLogix Corporation

©Copyright 2018-2022 NuraLogix Corporation

Chapter 8

File Documentation

8.1 ChunkData.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_CHUNKDATA_H
00017 #define LIBDFX_CHUNKDATA_H
00018
00019 #include "ChunkPayload.h"
00020
00021 namespace dfx {
00022
00035 class ChunkData {
00036   public:
00040     virtual ~ChunkData() {
00041         // Empty but necessary for vtable
00042     }
00043
00054     virtual ChunkPayload getChunkPayload() = 0;
00055 };
00056
00057 } // namespace dfx
00058
00059 #endif // LIBDFX_CHUNKDATA_H
```

8.2 ChunkPayload.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_CHUNKPAYLOAD_H
```

```
00017 #define LIBDFX_CHUNKPAYLOAD_H
00018
00019 #include <stdint.h>
00020 #include <vector>
00021
00022 namespace dfx {
00023
00029 struct ChunkPayload {
00031     uint8_t   valid;
00032
00034     uint32_t  startFrame;
00035
00037     uint32_t  endFrame;
00038
00040     uint32_t chunkNumber;
00041
00043     uint32_t numberChunks;
00044
00046     uint32_t firstChunkStartTime_s;
00047
00049     uint32_t startTime_s;
00050
00052     uint32_t endTime_s;
00053
00055     float    duration_s;
00056
00058     std::vector<uint8_t> payload;
00059
00061     std::vector<uint8_t> metadata;
00062 };
00063
00064 } // namespace dfx
00065
00066 #endif // LIBDFX_CHUNKPAYLOAD_H
```

8.3 Collector.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_COLLECTOR_H
00017 #define LIBDFX_COLLECTOR_H
00018
00019 #include "dfx/ChunkData.h"
00020 #include "dfx/Face.h"
00021 #include "dfx/Frame.h"
00022 #include "dfx/MeasurementResult.h"
00023
00024 #include <functional>
00025 #include <map>
00026 #include <memory>
00027 #include <string>
00028 #include <vector>
00029
00034 namespace dfx {
00035
00036 class Frame;
00037
00041 enum class ConstraintResult : char {
00042     Good = 0,
00043     Warn = 1,
00044     Error = 2
00045 };
00046
00053 enum class CollectorState : char {
00054     Waiting = 0,
00055     Collecting = 1,
00056     ChunkReady = 2,
00057     Completed = 3,
00058     Error = 4,
00059 };
```

©Copyright 2018-2022 NuraLogix Corporation

```
00060
00070 class Collector {
00071   public:
00075     virtual ~Collector() {
00076         // Empty but necessary for vtable
00077     }
00078
00086     virtual std::shared_ptr<dfx::Frame> createFrame(const dfx::VideoFrame &videoFrame) = 0;
00087
00103     virtual CollectorState startCollection() = 0;
00104
00116     virtual bool prepareMeasurement(const std::vector<uint8_t>& createMeasurementPayload) = 0;
00117
00130     virtual CollectorState getCollectorState() = 0;
00131
00140     virtual std::string getMode() const = 0;
00141
00162     virtual bool setTargetFPS(float targetFPS) = 0;
00163
00176     virtual bool setChunkDurationSeconds(float chunkDurationSeconds) = 0;
00177
00183     virtual float getChunkDurationSeconds() = 0;
00184
00193     virtual bool setNumberChunks(uint32_t numberChunks) = 0;
00194
00201     virtual uint32_t getNumberChunks() = 0;
00202
00216     virtual bool enableConstraint(const std::string &constraintID) = 0;
00217
00228     virtual bool disableConstraint(const std::string &constraintID) = 0;
00229
00235     virtual std::vector<std::string> getEnabledConstraints() = 0;
00236
00245     virtual std::vector<std::string> getAvailableConstraints() const = 0;
00246
00259     virtual ConstraintResult checkConstraints(std::shared_ptr<dfx::Frame> &frame,
      std::vector<std::pair<std::string,dfx::ConstraintResult» &results) = 0;
00260
00271     virtual std::string getConstraintErrorMessage(const std::string &violationID) = 0;
00272
00284     virtual bool setConstraintsConfig(const std::string& key, const std::string &propertyValue) = 0;
00285
00296     virtual std::string getConstraintsConfig(const std::string& key) = 0;
00297
00307     virtual std::vector<std::string> getRequiredPosePointIDs() const = 0;
00308
00321     virtual CollectorState defineRegions(std::shared_ptr<dfx::Frame> &frame) = 0;
00322
00334     virtual CollectorState extractChannels(std::shared_ptr<dfx::Frame> &frame) = 0;
00335
00342     virtual bool isChunkReady() const = 0;
00343
00362     virtual int32_t numberFramesNeeded() const = 0;
00363
00377     virtual std::shared_ptr<dfx::ChunkData> getChunkData() = 0;
00378
00389     virtual std::shared_ptr<dfx::MeasurementResult> decodeMeasurementResult(const
      std::vector<uint8_t>& measurementResponse) = 0;
00390
00406     virtual bool setProperty(const std::string &key, const std::string &value) = 0;
00407
00417     virtual std::string getProperty(const std::string& key) = 0;
00418
00422     virtual void removeProperty(const std::string& key) = 0;
00423
00433     virtual std::map<std::string, std::string> getProperties() = 0;
00434
00441     virtual std::string getLastErrorMessage() = 0;
00442
00451     virtual void cancelCollection() = 0;
00452
00466     virtual void stopCollection() = 0;
00467
00478     virtual void forceComplete() = 0;
00479
00489     virtual void resetCollection() = 0;
00490
00499     virtual bool setFaceAttribute(const std::string& faceID, FaceAttribute key, FaceAttributeValue
      value) = 0;
00500
00509     virtual bool setFaceAttribute(const std::string& faceID, FaceAttribute key, double value) = 0;
00510
00520     virtual bool setCloudResultsFeedback(const std::string& jsonCloudResults) = 0;
00521
00529     virtual bool initializeModel(int8_t modelType, const std::vector<unsigned char> &data) = 0;
00530 };
00531 } // namespace dfx
```

©Copyright 2018-2022 NuraLogix Corporation

```
00532
00533 #endif // LIBDFX_COLLECTOR_H
```

8.4 core.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_CPP_CORE_H
00017 #define LIBDFX_CPP_CORE_H
00018
00019 #include "dfxc/core.h"
00020
00021 #include "dfx/DFXFactory.h"
00022 #include "dfx/DFXErrors.h"
00023 #include "dfx/Face.h"
00024 #include "dfx/Frame.h"
00025 #include "dfx/Collector.h"
00026 #include "dfx/PosePoint.h"
00027 #include "dfx/MeasurementData.h"
00028 #include "dfx/MeasurementResult.h"
00029 #include "dfx/VideoFrame.h"
00030
00031 #endif
```

8.5 DFXErrors.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_ERRORS_H
00017 #define LIBDFX_ERRORS_H
00018
00019 #include "dfxc/dfx_error.h"
00020 #include <stdexcept>
00021
00029 namespace dfx {
00030
00034 struct Error {
00035     Error()
00036     : opaque(nullptr)
00037     {
00038     }
00039
00040     ~Error()
00041     {
00042         if (opaque) {
00043             dfx_error_destroy(opaque);
00044         }
00045     }
00046
00047     dfx_error* opaque;
00048 };
00049
```

©Copyright 2018-2022 NuraLogix Corporation

```
00053 class ThrowOnError {
00054 public:
00055     ~ThrowOnError() noexcept(false)
00056     {
00057         if (_error.opaque) {
00058             throw std::runtime_error(dfx_error_message(_error.opaque));
00059         }
00060     }
00061
00062     operator dfx_error**() { return &_error.opaque; }
00063
00064 private:
00065     struct Error _error;
00066 };
00067
00068 #ifdef DFX_DEBUG_DESKTOP
00069 #include <iostream>
00070 class LogOnError {
00071 public:
00072     LogOnError(std::ostream& os) : _os(os) {}
00073     ~LogOnError()
00074     {
00075         if (_error.opaque) {
00076             _os « _error.opaque « std::endl;
00077         }
00078     }
00079
00080     operator dfx_error**() { return &_error.opaque; }
00081
00082 private:
00083     std::ostream& _os;
00084     struct Error _error;
00085 };
00086 #endif
00087
00088 } // namespace dfx
00089
00090 #endif
```

8.6 DFXFactory.h

```
00001  /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_FACTORY_H
00017 #define LIBDFX_FACTORY_H
00018
00019 #include <map>
00020 #include <memory>
00021 #include <string>
00022
00023 #include "dfx/Collector.h"
00024
00025 namespace dfx {
00114 class DFXFactory {
00115   public:
00119     virtual ~DFXFactory() {
00120         // Empty but necessary for vtable
00121     }
00122
00128     virtual std::string getVersion() const = 0;
00129
00139     virtual std::string getSdkID() const = 0;
00140
00152     virtual bool setMode(const std::string& mode) = 0;
00153
00160     virtual std::string getMode() const = 0;
00161
00168     virtual std::string getLastErrorMessage() = 0;
00169
00179     virtual void setProperty(const std::string &key, const std::string &value) = 0;
```

©Copyright 2018-2022 NuraLogix Corporation

```
00180
00187     virtual std::string getProperty(const std::string &key) = 0;
00188
00196     virtual void removeProperty(const std::string &key) = 0;
00197
00205     virtual std::vector<std::string> getValidProperties() const = 0;
00206
00222     virtual bool initializeStudy(const std::vector<unsigned char> &data) = 0;
00223
00242     virtual bool addToStudy(const std::vector<unsigned char> &data) = 0;
00243
00263     virtual bool initializeStudyFromFile(const std::string &pathToFile) = 0;
00264
00283     virtual bool addToStudyFromFile(const std::string &pathToFile) = 0;
00284
00291     virtual std::shared_ptr<dfx::Collector> createCollector() = 0;
00292 };
00293 } // namespace dfx
00294
00295 #endif // LIBDFX_FACTORY_H
```

8.7 Face.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_FACE_H
00017 #define LIBDFX_FACE_H
00018
00019 #include "dfx/PosePoint.h"
00020
00021 #include <map>
00022 #include <string>
00023
00024 #include <opencv2/opencv.hpp>   // for core
00025
00026 namespace dfx {
00027     enum class FaceAttribute {
00028         SEX_ASSIGNED_AT_BIRTH = 1,        // SEX_ASSIGNED_MALE_AT_BIRTH, SEX_ASSIGNED_FEMALE_AT_BIRTH,
      SEX_NOT_PROVIDED
00029         AGE_YEARS = 2,                    // <NUMBER> of years old
00030         HEIGHT_CM = 3,                    // <NUMBER> of centimeters tall
00031         WEIGHT_KG = 4,                    // <NUMBER> of kilograms
00032         SMOKER = 5,                       // True (1) or False (0)
00033         HYPERTENSIVE = 6,                 // True (1) or False (0)
00034         BLOOD_PRESSURE_MEDICATION = 7,    // True (1) or False (0)
00035         DIABETES = 8,                     // DIABETES_NONE, DIABETES_TYPE1, DIABETES_TYPE2
00036     };
00037
00038     enum class FaceAttributeValue {
00039         SEX_NOT_PROVIDED = 1,
00040         SEX_ASSIGNED_MALE_AT_BIRTH = 2,
00041         SEX_ASSIGNED_FEMALE_AT_BIRTH = 3,
00042
00043         DIABETES_NONE = 4,
00044         DIABETES_TYPE1 = 5,
00045         DIABETES_TYPE2 = 6,
00046     };
00047
00058 struct Face {
00067     std::string id;
00068
00077     cv::Rect faceRect;
00078
00086     bool poseValid;
00087
00094     bool detected;
00095
00102     std::map<std::string, dfx::PosePoint> posePoints;
00103
00119     std::map<std::string, double> attributes;
```

©Copyright 2018-2022 NuraLogix Corporation

```
00120 };
00121 } // namespace dfx
00122
00123 #endif // LIBDFX_FACE_H
```

8.8 Frame.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_FRAME_H
00017 #define LIBDFX_FRAME_H
00018
00019 #include "dfx/Face.h"
00020 #include "dfx/VideoFrame.h"
00021
00022 #include <memory>
00023 #include <string>
00024
00025 namespace dfx {
00037 class Frame {
00038   public:
00042     virtual ~Frame() {
00043         // Empty but necessary for vtable
00044     }
00045
00052     virtual VideoFrame getVideoFrame() const = 0;
00053
00063     virtual bool addFace(std::shared_ptr<dfx::Face> face) = 0;
00064
00080     virtual std::vector<float> getQualityMetrics(float iso, float exposure) = 0;
00081
00087     virtual std::vector<float> getOpticalQualityMetrics() = 0;
00088
00100     virtual int getStarRating(std::string& feedback) = 0;
00101
00110     virtual int getOpticalQualityRating(std::string& feedback) = 0;
00111
00122     virtual bool addMarker(const std::string &label) = 0;
00123
00129     virtual std::vector<std::string> getMarkers() const = 0;
00130
00145     virtual std::vector<std::string> getDesiredFaceAttributes() const = 0;
00146
00162     virtual std::vector<std::string> getDesiredDeviceAttributes() const = 0;
00163
00170     virtual double getDeviceAttribute(const std::string& key) const = 0;
00171
00183     virtual void setDeviceAttribute(const std::string& key, double value) = 0;
00184
00193     virtual std::vector<std::string> getFaceIdentifiers() const = 0;
00194
00204     virtual std::vector<std::string> getRegionNames(const std::string &faceID) const = 0;
00205
00214     virtual std::vector<cv::Point> getRegionPolygon(const std::string &faceID, const std::string
      &regionName) const = 0;
00215
00225     virtual int32_t getRegionIntProperty(const std::string& faceID, const std::string& regionName,
      const std::string& property) const = 0;
00226
00235     virtual cv::Mat getRegionHistogram(const std::string &faceID, const std::string &regionName) const
      = 0;
00236 };
00237 } // namespace dfx
00238
00239 #endif // LIBDFX_FRAME_H
```

©Copyright 2018-2022 NuraLogix Corporation

8.9 MeasurementData.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_MEASUREMENTDATA_H
00017 #define LIBDFX_MEASUREMENTDATA_H
00018
00019 #include <stdint.h>
00020 #include <string>
00021 #include <vector>
00022
00023 namespace dfx {
00024
00030 class MeasurementData {
00031 public:
00032     /*
00033      * \brief MeasurementData destructor
00034      */
00035     virtual ~MeasurementData() {
00036         // Empty but necessary for vtable
00037     }
00038
00045     virtual std::vector<std::string> getDataPropertyKeys() = 0;
00046
00058     virtual std::string getDataProperty(const std::string& key) = 0;
00059
00072     virtual std::vector<double> getData() = 0;
00073 };
00074
00075 } // namespace dfx
00076
00077 #endif // LIBDFX_MEASUREMENTDATA_H
```

8.10 MeasurementResult.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_MEASUREMENTRESULT_H
00017 #define LIBDFX_MEASUREMENTRESULT_H
00018
00019 #include "dfx/MeasurementData.h"
00020
00021 #include <memory>
00022 #include <stdint.h>
00023 #include <string>
00024 #include <vector>
00025
00026 namespace dfx {
00027
00035 class MeasurementResult {
00036 public:
00037     /*
00038      * \brief MeasurementResult destructor
00039      */
00040     virtual ~MeasurementResult() {
00041         // Empty but necessary for vtable
```

©Copyright 2018-2022 NuraLogix Corporation

```
00042     }
00043
00049     virtual uint8_t isValid() = 0;
00050
00057     virtual std::vector<std::string> getMeasurementPropertyKeys() = 0;
00058
00069     virtual std::string getMeasurementProperty(const std::string& key) = 0;
00070
00084     virtual std::vector<std::string> getMeasurementDataKeys() = 0;
00085
00094     virtual std::shared_ptr<MeasurementData> getMeasurementData(const std::string& key) = 0;
00095
00104     virtual std::string getErrorCode() = 0;
00105 };
00106
00107 } // namespace dfx
00108
00109 #endif // LIBDFX_MEASUREMENTRESULT_H
```

8.11 PosePoint.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_POSEPOINT_H
00017 #define LIBDFX_POSEPOINT_H
00018
00019 #include <opencv2/opencv.hpp> // For cv::Point3f (an cv::Mat) from core
00020
00021 namespace dfx {
00022
00037 struct PosePoint {
00045     cv::Point3f point;
00046
00054     bool valid;
00055
00066     bool estimated;
00067
00075     float quality;
00076 };
00077 } // namespace dfx
00078
00079 #endif // LIBDFX_POSEPOINT_H
```

8.12 VideoFrame.h

```
00001 /*
00002  *              Copyright (c) 2016-2019, Nuralogix Corp.
00003  *                      All Rights reserved
00004  *
00005  *      THIS SOFTWARE IS LICENSED BY AND IS THE CONFIDENTIAL AND
00006  *      PROPRIETARY PROPERTY OF NURALOGIX CORP. IT IS
00007  *      PROTECTED UNDER THE COPYRIGHT LAWS OF THE USA, CANADA
00008  *      AND OTHER FOREIGN COUNTRIES. THIS SOFTWARE OR ANY
00009  *      PART THEREOF, SHALL NOT, WITHOUT THE PRIOR WRITTEN CONSENT
00010  *      OF NURALOGIX CORP, BE USED, COPIED, DISCLOSED,
00011  *      DECOMPILED, DISASSEMBLED, MODIFIED OR OTHERWISE TRANSFERRED
00012  *      EXCEPT IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF A
00013  *      NURALOGIX CORP SOFTWARE LICENSE AGREEMENT.
00014  */
00015 #pragma once
00016 #ifndef LIBDFX_VIDEOFRAME_H
00017 #define LIBDFX_VIDEOFRAME_H
00018
00019 #include <limits>
00020 #include <opencv2/opencv.hpp>  // For cv::Mat (and cv::Point3f) from core
```

©Copyright 2018-2022 NuraLogix Corporation

```
00021
00022 namespace dfx {
00023
00035 struct VideoFrame {
00047     enum class ChannelOrder {
00049         BGR = 1,
00050
00052         RGB = 2,
00053
00055         BGRA = 3,
00056
00058         RGBA = 4,
00059
00061         Infrared = 5,
00062
00064         Infrared888 = 6,
00065
00067         BGR_Infrared = 7,
00068
00070         RGB_Infrared = 8,
00071
00073         Gray = 9
00074     };
00075
00082     cv::Mat image;
00083
00091     ChannelOrder order;
00092
00104     uint64_t timestamp_ns;
00105
00116     uint32_t number;
00117
00145     double timestamp_millisec{std::numeric_limits<double>::quiet_NaN()};
00146 };
00147 } // namespace dfx
00148
00149 #endif // LIBDFX_VIDEOFRAME_H
```

©Copyright 2018-2022 NuraLogix Corporation

Index

addFace
    dfx::Frame, 41
addMarker
    dfx::Frame, 41
addToStudy
    dfx::DFXFactory, 32
addToStudyFromFile
    dfx::DFXFactory, 33
attributes
    dfx::Face, 38

BGR
    dfx::VideoFrame, 53
BGR_Infrared
    dfx::VideoFrame, 53
BGRA
    dfx::VideoFrame, 53

cancelCollection
    dfx::Collector, 18
ChannelOrder
    dfx::VideoFrame, 53
checkConstraints
    dfx::Collector, 18
ChunkReady
    dfx, 13
Collecting
    dfx, 13
CollectorState
    dfx, 12
Completed
    dfx, 13
ConstraintResult
    dfx, 13
createCollector
    dfx::DFXFactory, 33
createFrame
    dfx::Collector, 19

decodeMeasurementResult
    dfx::Collector, 19
defineRegions
    dfx::Collector, 20
Deprecated List, 3
detected
    dfx::Face, 38
dfx, 11
    ChunkReady, 13
    Collecting, 13
    CollectorState, 12
    Completed, 13
    ConstraintResult, 13
    Error, 13
    Good, 13
    Waiting, 13
    Warn, 13
DFX Extraction library v4.15.8, 1
dfx/ChunkData.h, 57
dfx/ChunkPayload.h, 57
dfx/Collector.h, 58
dfx/core.h, 60
dfx/DFXErrors.h, 60
dfx/DFXFactory.h, 61
dfx/Face.h, 62
dfx/Frame.h, 63
dfx/MeasurementData.h, 64
dfx/MeasurementResult.h, 64
dfx/PosePoint.h, 65
dfx/VideoFrame.h, 65
dfx::ChunkData, 15
    getChunkPayload, 15
dfx::ChunkPayload, 16
dfx::Collector, 16
    cancelCollection, 18
    checkConstraints, 18
    createFrame, 19
    decodeMeasurementResult, 19
    defineRegions, 20
    disableConstraint, 20
    enableConstraint, 20
    extractChannels, 21
    forceComplete, 21
    getAvailableConstraints, 21
    getChunkData, 22
    getChunkDurationSeconds, 22
    getCollectorState, 22
    getConstraintErrorMessage, 22
    getConstraintsConfig, 23
    getEnabledConstraints, 23
    getLastErrorMessage, 23
    getMode, 24
    getNumberChunks, 24
    getProperties, 24
    getProperty, 24
    getRequiredPosePointIDs, 25
    initializeModel, 25
    isChunkReady, 25
    numberFramesNeeded, 25
    prepareMeasurement, 26

resetCollection, 26
setChunkDurationSeconds, 26
setCloudResultsFeedback, 27
setConstraintsConfig, 27
setFaceAttribute, 28
setNumberChunks, 28
setProperty, 29
setTargetFPS, 29
startCollection, 30
stopCollection, 30
dfx::DFXFactory, 30
addToStudy, 32
addToStudyFromFile, 33
createCollector, 33
getLastErrorMessage, 33
getMode, 34
getProperty, 34
getSdkID, 34
getValidProperties, 34
getVersion, 35
initializeStudy, 35
initializeStudyFromFile, 35
removeProperty, 36
setMode, 36
setProperty, 37
dfx::Error, 37
dfx::Face, 38
attributes, 38
detected, 38
faceRect, 39
id, 39
posePoints, 39
poseValid, 39
dfx::Frame, 40
addFace, 41
addMarker, 41
getDesiredDeviceAttributes, 41
getDesiredFaceAttributes, 42
getDeviceAttribute, 42
getFaceIdentifiers, 42
getMarkers, 43
getOpticalQualityMetrics, 43
getOpticalQualityRating, 43
getQualityMetrics, 43
getRegionHistogram, 44
getRegionIntProperty, 44
getRegionNames, 45
getRegionPolygon, 45
getStarRating, 45
getVideoFrame, 46
setDeviceAttribute, 46
dfx::MeasurementData, 47
getData, 47
getDataProperty, 47
getDataPropertyKeys, 47
dfx::MeasurementResult, 48
getErrorCode, 48
getMeasurementData, 48
getMeasurementDataKeys, 49
getMeasurementProperty, 49
getMeasurementPropertyKeys, 49
isValid, 50
dfx::PosePoint, 50
estimated, 51
point, 51
quality, 51
valid, 52
dfx::ThrowOnError, 52
dfx::VideoFrame, 52
BGR, 53
BGR_Infrared, 53
BGRA, 53
ChannelOrder, 53
Gray, 53
image, 54
Infrared, 53
Infrared888, 53
number, 54
order, 54
RGB, 53
RGB_Infrared, 53
RGBA, 53
timestamp_millisec, 54
timestamp_ns, 54
disableConstraint
dfx::Collector, 20

enableConstraint
dfx::Collector, 20
Error
dfx, 13
estimated
dfx::PosePoint, 51
extractChannels
dfx::Collector, 21

faceRect
dfx::Face, 39
forceComplete
dfx::Collector, 21

getAvailableConstraints
dfx::Collector, 21
getChunkData
dfx::Collector, 22
getChunkDurationSeconds
dfx::Collector, 22
getChunkPayload
dfx::ChunkData, 15
getCollectorState
dfx::Collector, 22
getConstraintErrorMessage
dfx::Collector, 22
getConstraintsConfig
dfx::Collector, 23
getData
dfx::MeasurementData, 47

©Copyright 2018-2022 NuraLogix Corporation

getDataProperty
dfx::MeasurementData, 47
getDataPropertyKeys
dfx::MeasurementData, 47
getDesiredDeviceAttributes
dfx::Frame, 41
getDesiredFaceAttributes
dfx::Frame, 42
getDeviceAttribute
dfx::Frame, 42
getEnabledConstraints
dfx::Collector, 23
getErrorCode
dfx::MeasurementResult, 48
getFaceIdentifiers
dfx::Frame, 42
getLastErrorMessage
dfx::Collector, 23
dfx::DFXFactory, 33
getMarkers
dfx::Frame, 43
getMeasurementData
dfx::MeasurementResult, 48
getMeasurementDataKeys
dfx::MeasurementResult, 49
getMeasurementProperty
dfx::MeasurementResult, 49
getMeasurementPropertyKeys
dfx::MeasurementResult, 49
getMode
dfx::Collector, 24
dfx::DFXFactory, 34
getNumberChunks
dfx::Collector, 24
getOpticalQualityMetrics
dfx::Frame, 43
getOpticalQualityRating
dfx::Frame, 43
getProperties
dfx::Collector, 24
getProperty
dfx::Collector, 24
dfx::DFXFactory, 34
getQualityMetrics
dfx::Frame, 43
getRegionHistogram
dfx::Frame, 44
getRegionIntProperty
dfx::Frame, 44
getRegionNames
dfx::Frame, 45
getRegionPolygon
dfx::Frame, 45
getRequiredPosePointIDs
dfx::Collector, 25
getSdkID
dfx::DFXFactory, 34
getStarRating
dfx::Frame, 45
getValidProperties
dfx::DFXFactory, 34
getVersion
dfx::DFXFactory, 35
getVideoFrame
dfx::Frame, 46
Good
dfx, 13
Gray
dfx::VideoFrame, 53

id
dfx::Face, 39
image
dfx::VideoFrame, 54
Infrared
dfx::VideoFrame, 53
Infrared888
dfx::VideoFrame, 53
initializeModel
dfx::Collector, 25
initializeStudy
dfx::DFXFactory, 35
initializeStudyFromFile
dfx::DFXFactory, 35
isChunkReady
dfx::Collector, 25
isValid
dfx::MeasurementResult, 50

number
dfx::VideoFrame, 54
numberFramesNeeded
dfx::Collector, 25

order
dfx::VideoFrame, 54

point
dfx::PosePoint, 51
posePoints
dfx::Face, 39
poseValid
dfx::Face, 39
prepareMeasurement
dfx::Collector, 26

quality
dfx::PosePoint, 51

removeProperty
dfx::DFXFactory, 36
resetCollection
dfx::Collector, 26
RGB
dfx::VideoFrame, 53
RGB_Infrared
dfx::VideoFrame, 53
RGBA

© Copyright 2018-2022 NuraLogix Corporation

dfx::VideoFrame, 53

setChunkDurationSeconds
dfx::Collector, 26
setCloudResultsFeedback
dfx::Collector, 27
setConstraintsConfig
dfx::Collector, 27
setDeviceAttribute
dfx::Frame, 46
setFaceAttribute
dfx::Collector, 28
setMode
dfx::DFXFactory, 36
setNumberChunks
dfx::Collector, 28
setProperty
dfx::Collector, 29
dfx::DFXFactory, 37
setTargetFPS
dfx::Collector, 29
startCollection
dfx::Collector, 30
stopCollection
dfx::Collector, 30

timestamp_millisec
dfx::VideoFrame, 54
timestamp_ns
dfx::VideoFrame, 54

valid
dfx::PosePoint, 52

Waiting
dfx, 13
Warn
dfx, 13

©Copyright 2018-2022 NuraLogix Corporation