



DFX Extraction Library v4.12.0

NuraLogix Corporation

Contents

1 Main Page	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 Namespace Documentation	9
5.1 dfx Namespace Reference	9
5.1.1 Detailed Description	10
5.1.2 Enumeration Type Documentation	10
5.1.2.1 CollectorState	10
5.1.2.2 ConstraintResult	11
6 Class Documentation	13
6.1 dfx::ChunkData Class Reference	13
6.1.1 Detailed Description	13
6.1.2 Member Function Documentation	13
6.1.2.1 getChunkPayload()	14
6.2 dfx::ChunkPayload Struct Reference	14
6.2.1 Detailed Description	15
6.3 dfx::Collector Class Reference	15
6.3.1 Detailed Description	17
6.3.2 Member Function Documentation	17
6.3.2.1 cancelCollection()	17
6.3.2.2 checkConstraints()	17
6.3.2.3 createFrame()	17
6.3.2.4 decodeMeasurementResult()	18
6.3.2.5 defineRegions()	18
6.3.2.6 disableConstraint()	19
6.3.2.7 enableConstraint()	19
6.3.2.8 extractChannels()	19
6.3.2.9 forceComplete()	20
6.3.2.10 getAvailableConstraints()	20
6.3.2.11 getChunkData()	20
6.3.2.12 getChunkDurationSeconds()	21
6.3.2.13 getCollectorState()	21
6.3.2.14 getConstraintErrorMessage()	21
6.3.2.15 getConstraintsConfig()	22
6.3.2.16 getEnabledConstraints()	22
6.3.2.17 getLastErrorMessage()	22
6.3.2.18 getMode()	22

6.3.2.19	getNumberChunks()	23
6.3.2.20	getProperties()	23
6.3.2.21	getProperty()	23
6.3.2.22	getRequiredPosePointIDs()	23
6.3.2.23	initializeModel()	23
6.3.2.24	isChunkReady()	24
6.3.2.25	numberFramesNeeded()	24
6.3.2.26	prepareMeasurement()	24
6.3.2.27	resetCollection()	25
6.3.2.28	setChunkDurationSeconds()	25
6.3.2.29	setCloudResultsFeedback()	25
6.3.2.30	setConstraintsConfig()	26
6.3.2.31	setFaceAttribute() [1/2]	26
6.3.2.32	setFaceAttribute() [2/2]	27
6.3.2.33	setNumberChunks()	27
6.3.2.34	setProperty()	27
6.3.2.35	setTargetFPS()	28
6.3.2.36	startCollection()	28
6.3.2.37	stopCollection()	29
6.4	dfx::DFXFactory Class Reference	29
6.4.1	Detailed Description	30
6.4.2	Member Function Documentation	31
6.4.2.1	addToStudy()	31
6.4.2.2	addToStudyFromFile()	31
6.4.2.3	createCollector()	32
6.4.2.4	getLastErrorMessage()	32
6.4.2.5	getMode()	32
6.4.2.6	getProperty()	32
6.4.2.7	getSdkID()	33
6.4.2.8	getValidProperties()	33
6.4.2.9	getVersion()	34
6.4.2.10	initializeStudy()	34
6.4.2.11	initializeStudyFromFile()	35
6.4.2.12	removeProperty()	35
6.4.2.13	setMode()	36
6.4.2.14	setProperty()	36
6.5	dfx::Error Struct Reference	37
6.6	dfx::Face Struct Reference	37
6.6.1	Detailed Description	37
6.6.2	Member Data Documentation	37
6.6.2.1	attributes	38
6.6.2.2	detected	38
6.6.2.3	faceRect	38
6.6.2.4	id	38
6.6.2.5	posePoints	38
6.6.2.6	poseValid	39
6.7	dfx::Frame Class Reference	39
6.7.1	Detailed Description	40
6.7.2	Member Function Documentation	40
6.7.2.1	addFace()	40
6.7.2.2	addMarker()	40
6.7.2.3	getDesiredDeviceAttributes()	41
6.7.2.4	getDesiredFaceAttributes()	41
6.7.2.5	getDeviceAttribute()	41
6.7.2.6	getFacelIdentifiers()	42
6.7.2.7	getMarkers()	42
6.7.2.8	getOpticalQualityMetrics()	42
6.7.2.9	getOpticalQualityRating()	42

6.7.2.10	getQualityMetrics()	43
6.7.2.11	getRegionHistogram()	43
6.7.2.12	getRegionIntProperty()	44
6.7.2.13	getRegionNames()	44
6.7.2.14	getRegionPolygon()	45
6.7.2.15	getStarRating()	45
6.7.2.16	getVideoFrame()	45
6.7.2.17	setDeviceAttribute()	46
6.8	dfx::MeasurementData Class Reference	46
6.8.1	Detailed Description	46
6.8.2	Member Function Documentation	46
6.8.2.1	getData()	47
6.8.2.2	getDataProperty()	47
6.8.2.3	getDataPropertyKeys()	47
6.9	dfx::MeasurementResult Class Reference	48
6.9.1	Detailed Description	48
6.9.2	Member Function Documentation	48
6.9.2.1	getErrorCode()	48
6.9.2.2	getMeasurementData()	48
6.9.2.3	getMeasurementDataKeys()	49
6.9.2.4	getMeasurementProperty()	49
6.9.2.5	getMeasurementPropertyKeys()	50
6.9.2.6	isValid()	50
6.10	dfx::PosePoint Struct Reference	50
6.10.1	Detailed Description	51
6.10.2	Member Data Documentation	51
6.10.2.1	estimated	51
6.10.2.2	point	52
6.10.2.3	quality	52
6.10.2.4	valid	52
6.11	dfx::ThrowOnError Class Reference	52
6.12	dfx::VideoFrame Struct Reference	52
6.12.1	Detailed Description	53
6.12.2	Member Enumeration Documentation	53
6.12.2.1	ChannelOrder	53
6.12.3	Member Data Documentation	54
6.12.3.1	image	54
6.12.3.2	number	54
6.12.3.3	order	54
6.12.3.4	timestamp_millisec	54
6.12.3.5	timestamp_ns	55

Chapter 1

Main Page

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::Collectors`.

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::Frames` (created from `dfx::VideoFrames`) have accurate timestamps and `dfx::Faces`. 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::Frames` 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.

Subject to change

Chapter 2

Deprecated List

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 std::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 std::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 std::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`

Subject to change

Chapter 3

Namespace Index

3.1 Namespace List

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

dfx	9
-------------------------------	---

Subject to change

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	13
dfx::ChunkPayload	Chunk that was captured by a Collection	14
dfx::Collector	Collector extracts region channel information from video frames for server signal processing	15
dfx::DFXFactory	DFXFactory is the primary entry point for the collector and is used create a collector	29
dfx::Error	37
dfx::Face	Identifies the properties associated with faces within the video frame	37
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	39
dfx::MeasurementData	46
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

Subject to change

Chapter 5

Namespace Documentation

5.1 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 [ConstraintResult](#) : char { [ConstraintResult::Good](#) = 0, [ConstraintResult::Warn](#) = 1, [ConstraintResult::Error](#) = 2 }
The possible constraint return values.

- enum [CollectorState](#) : char {
 CollectorState::Waiting = 0, CollectorState::Collecting = 1, CollectorState::ChunkReady = 2, CollectorState::Completed = 3,
 CollectorState::Error = 4 }

A [Collector](#) state indicates whether a [Collector](#) is in error or ready to collect data.

- enum [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 [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 }

5.1.1 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

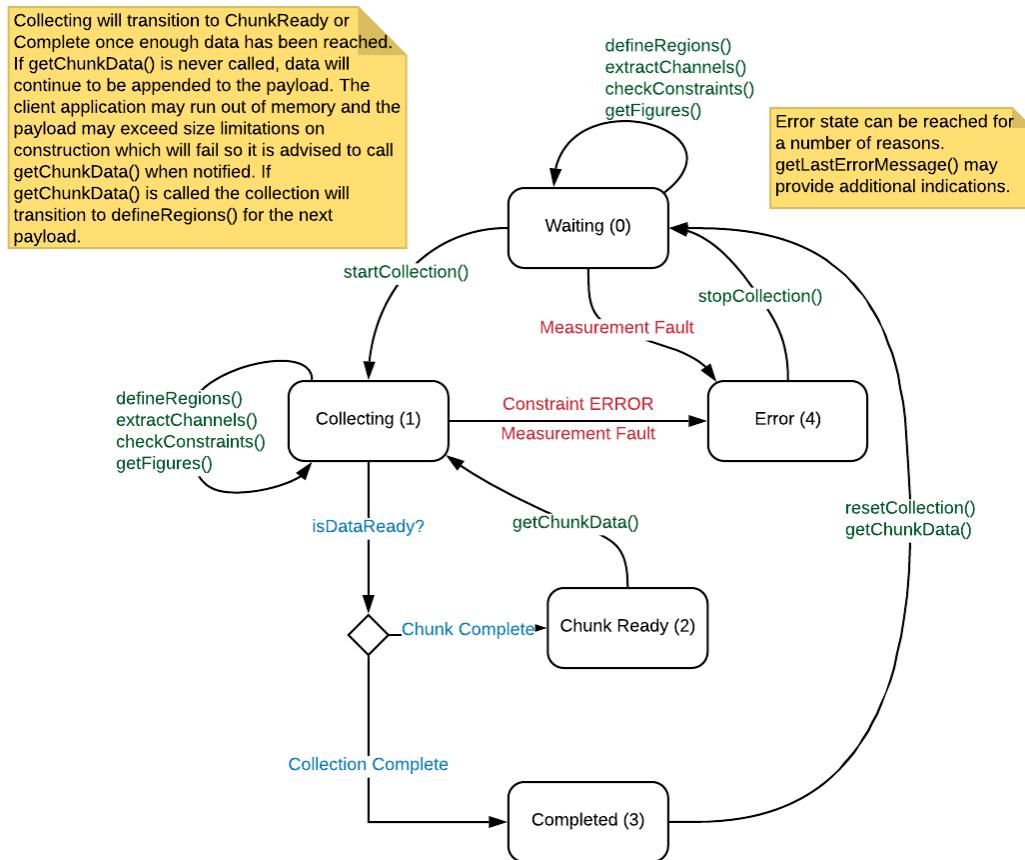
"Hourglass Interfaces for C++ APIs"

5.1.2 Enumeration Type Documentation

5.1.2.1 CollectorState

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

A [Collector](#) state indicates whether a [Collector](#) is in error or ready to collect data.

**Enumerator**

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

5.1.2.2 ConstraintResult

```
enum 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

Subject to change

Chapter 6

Class Documentation

6.1 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.

6.1.1 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.

6.1.2 Member Function Documentation

6.1.2.1 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

6.2 dfx::ChunkPayload Struct Reference

Chunk that was captured by a Collection.

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

Public Attributes

- **uint8_t valid**
if this payload was successfully 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)

6.2.1 Detailed Description

Chunk that was captured by a Collection.

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

- dfx/ChunkPayload.h

6.3 dfx::Collector Class Reference

[Collector](#) extracts region channel information from video frames for server signal processing.

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

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](#) () const =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

- virtual std::string **getConstraintErrorMessage** (const std::string &violationID)=0

Check for constraint violations.
- virtual bool **setConstraintsConfig** (const std::string &key, const std::string &propertyValue)=0

Obtain a message which represents the violation ID.

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

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](#).

6.3.1 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.

6.3.2 Member Function Documentation

6.3.2.1 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.

6.3.2.2 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

<i>frame</i>	the video frame currently being operated on
<i>results</i>	is set of constraints and associated state

Returns

result of the constraint check

6.3.2.3 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

<i>videoFrame</i>	is the video Frame to wrap.
-------------------	---

Returns

a reference to a [Frame](#).

6.3.2.4 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](#).

Parameters

<i>measurementResponse</i>	from the server to decode
----------------------------	---------------------------

Returns

the [MeasurementResult](#) object

6.3.2.5 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

<i>frame</i>	the video frame currently being operated on
--------------	---

Returns

the [Collector](#) state after define regions

6.3.2.6 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

<i>constraint</i>	to disable
-------------------	------------

Returns

true if the constraintID was changeable, false otherwise.

6.3.2.7 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

<i>constraint</i>	to enable
-------------------	-----------

Returns

true if the constraintID was changeable, false otherwise.

6.3.2.8 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

<i>frame</i>	the video frame currently being operated on
--------------	---

Returns

the [Collector](#) state after extract channels

6.3.2.9 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.

6.3.2.10 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.

6.3.2.11 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.

6.3.2.12 getChunkDurationSeconds()

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

Returns the desired collected chunk duration in seconds.

Returns

the chunk durations in seconds.

6.3.2.13 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.

6.3.2.14 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 constraint message is a more meaningful representation of the constraint violation ID.

Parameters

<i>violationID</i>	the constraint ID to obtain the error message for
--------------------	---

Returns

error message of the corresponding violationID.

6.3.2.15 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

<i>key</i>	the configuration key to work on, currently "json"
------------	--

Returns

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

6.3.2.16 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.

6.3.2.17 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

6.3.2.18 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.

6.3.2.19 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.

6.3.2.20 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

6.3.2.21 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.

6.3.2.22 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.

6.3.2.23 initializeModel()

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

Configures a model within the [Collector](#).

Parameters

<i>modelType</i>	A numeric identifier associated with the model implementation
<i>data</i>	Configuration data associated with the model

Returns

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

6.3.2.24 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

6.3.2.25 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 monotonically 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.

6.3.2.26 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.

Parameters

<i>createMeasurementPayload</i>	the payload from DFX server
---------------------------------	-----------------------------

Returns

true if collector was happy, false otherwise

6.3.2.27 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()`.

6.3.2.28 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

<i>chunkDurationSeconds</i>	the chunk duration in seconds.
-----------------------------	--------------------------------

Returns

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

6.3.2.29 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

<i>jsonCloudResults</i>	the JSON results obtained from the Cloud API
-------------------------	--

Returns

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

6.3.2.30 setConstraintsConfig()

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

Configures the constraint system properties.

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

Parameters

<i>key</i>	the configuration key to work on, currently "json"
<i>property</i>	the value to set this property to

Returns

true if the property was successfully set, false on error

6.3.2.31 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

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

Returns

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

6.3.2.32 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

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

Returns

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

6.3.2.33 setNumberChunks()

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

Sets number of chunks before the measurement is complete.

Parameters

<i>numberChunks</i>	the number of chunks.
---------------------	-----------------------

Returns

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

6.3.2.34 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

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

Returns

true if property was set, false otherwise

6.3.2.35 [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 than a live extraction which needs to share CPU processing power for video frame handling with video frame processing.

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

Parameters

<i>targetFPS</i>	the anticipated target FPS
------------------	----------------------------

Returns

true if the targetFPS was successfully changed, false otherwise.

6.3.2.36 [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.

6.3.2.37 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

6.4 dfx::DFXFactory Class Reference

[DFXFactory](#) is the primary entry point for the collector and is used create a collector.

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

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 error 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

- **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.

6.4.1 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 relevant 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();
collector->setTargetFPS(30.0f);
collector->prepareMeasurement(measurementResponsePayload);
// Enable all possible client constraints 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();
}
}
}

```

6.4.2 Member Function Documentation

6.4.2.1 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.

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

<i>data</i>	to add to the study definition
-------------	--------------------------------

Returns

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

See also

[DFXNetworkAPI](#)

6.4.2.2 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

<i>pathToFile</i>	to add to the study definition
-------------------	--------------------------------

Returns

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

See also

DFXNetworkAPI

6.4.2.3 **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.

6.4.2.4 **getLastErrorMessage()**

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

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

Returns

the last known error

6.4.2.5 **getMode()**

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

Returns the current operating mode when creating collectors.

Returns

the currently configured operating mode

6.4.2.6 **getProperty()**

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

Returns a property value for a key.

Parameters

<i>key</i>	to obtain the property value for.
------------	-----------------------------------

Returns

the property if it exists or an empty string.

6.4.2.7 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.

6.4.2.8 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](#)

6.4.2.9 **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.

6.4.2.10 **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

<code>data</code>	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](#)

6.4.2.11 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

<code>pathToFile</code>	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](#)

6.4.2.12 removeProperty()

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

Removes a property from the factory if it exists.

Parameters

<i>key</i>	of property to remove if it exists, ignored if it does not exist.
------------	---

See also

[getValidProperties](#), [setProperties](#)

6.4.2.13 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

<i>mode</i>	the string representing the operating mode.
-------------	---

Returns

true if mode was successfully changed, false otherwise.

6.4.2.14 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

<i>key</i>	of property to configure.
<i>value</i>	of property to configure.

See also

[getValidProperties](#)

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

- dfx/DFXFactory.h

6.5 dfx::Error Struct Reference

Public Attributes

- dfx_error * **opaque**

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

- dfx/DFXErrors.h

6.6 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.

6.6.1 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".

6.6.2 Member Data Documentation

6.6.2.1 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.

6.6.2.2 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.

6.6.2.3 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.

6.6.2.4 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.

6.6.2.5 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.

6.6.2.6 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

6.7 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 > **getFacelidentifiers** () const =0

- `virtual std::vector< std::string > getRegionNames (const std::string &faceID) const =0`
obtains the known face identifiers within this frame.
- `virtual std::vector< cv::Point > getRegionPolygon (const std::string &faceID, const std::string ®ionName) const =0`
obtains the names of regions which are defined for the face within this frame.
- `virtual int32_t getRegionIntProperty (const std::string &faceID, const std::string ®ionName, const std::string &property) const =0`
obtain the polygon for the region of interest.
- `virtual cv::Mat getRegionHistogram (const std::string &faceID, const std::string ®ionName) const =0`
Regions can have numeric properties attached to them.
- `virtual cv::Mat getRegionHistogram (const std::string &faceID, const std::string ®ionName) const =0`
obtain the histogram for a region of interest.

6.7.1 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.

6.7.2 Member Function Documentation

6.7.2.1 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

<code>face</code>	data to add to this <code>Frame</code> .
-------------------	--

Returns

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

6.7.2.2 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 added and annotated against a frame.

Parameters

<i>label</i>	the string marker identifier to add to the frame.
--------------	---

Returns

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

6.7.2.3 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](#)

6.7.2.4 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

6.7.2.5 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

<i>key</i>	the attribute name.
------------	---------------------

Returns

value the value of the attribute or zero.

6.7.2.6 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.

6.7.2.7 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.

6.7.2.8 getOpticalQualityMetrics()

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

Calculates the optical quality metrics for the current frame.

Returns

the quality metrics

6.7.2.9 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

<i>the</i>	feedback string
------------	-----------------

Returns

the 5-star rating

See also

[getQualityMetrics](#)

6.7.2.10 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

<i>iso</i>	the current iso setting
<i>exposure</i>	the current exposure setting

Returns

the quality metrics

6.7.2.11 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

<i>faceID</i>	identifier for the face of interest.
<i>regionName</i>	which histogram is desired.

Returns

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

6.7.2.12 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

<i>faceID</i>	identifier for the face of interest.
<i>regionName</i>	to inspect the property of.
<i>property</i>	to be returned

Returns

the numeric value of the property.

6.7.2.13 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

<i>faceID</i>	identifier for the face of interest.
---------------	--------------------------------------

Returns

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

6.7.2.14 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

<i>faceID</i>	identifier for the face of interest.
<i>regionName</i>	which points are needed for drawing.

Returns

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

6.7.2.15 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

<i>the</i>	feedback string
------------	-----------------

Returns

the 5-star rating

See also

[getQualityMetrics](#)

6.7.2.16 getVideoFrame()

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

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

return the associated `VideoFrame`

6.7.2.17 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

<i>key</i>	the attribute name.
<i>value</i>	the value of the attribute.

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

- dfx/Frame.h

6.8 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 MeasurementData.

6.8.1 Detailed Description

A [MeasurementData](#) contains [MeasurementResult](#) data for a specific signal.

6.8.2 Member Function Documentation

6.8.2.1 getData()

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

Returns the numerical data associated with this MeasurementData.

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](#).

6.8.2.2 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.

6.8.2.3 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

6.9 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](#)

6.9.1 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.

6.9.2 Member Function Documentation

6.9.2.1 [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.

6.9.2.2 [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

<i>key</i>	to obtain the MeasurementData for
------------	---

Returns

the [MeasurementData](#)

6.9.2.3 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 measurement 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

6.9.2.4 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

<i>key</i>	the measurement property to look up
------------	-------------------------------------

Returns

the valid of the measurement property or an empty string

6.9.2.5 `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

6.9.2.6 `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

6.10 dfx::PosePoint Struct Reference

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

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

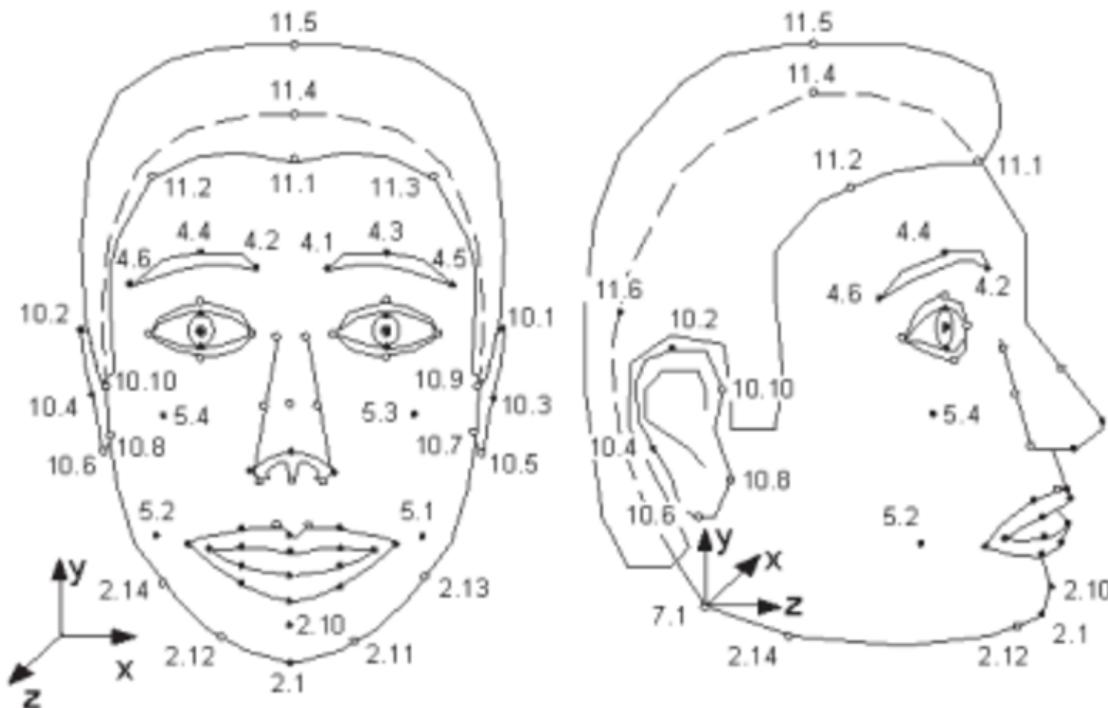
Public Attributes

- 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.

6.10.1 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](#)

6.10.2 Member Data Documentation

6.10.2.1 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.

6.10.2.2 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.

6.10.2.3 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.

6.10.2.4 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

6.11 dfx::ThrowOnError Class Reference

Public Member Functions

- **operator dfx_error ** ()**

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

- dfx/DFXErrors.h

6.12 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 `ChannelOrder` {
 `ChannelOrder::BGR` = 1, `ChannelOrder::RGB` = 2, `ChannelOrder::BGRA` = 3, `ChannelOrder::RGBA` = 4,
 `ChannelOrder::Infrared` = 5, `ChannelOrder::Infrared888` = 6, `ChannelOrder::BGR_Infrared` = 7, `ChannelOrder::RGB_Infrared` = 8,
 `ChannelOrder::Gray` = 9
 }

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

6.12.1 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 its position within the stream.

6.12.2 Member Enumeration Documentation

6.12.2.1 ChannelOrder

```
enum 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 grayscale

6.12.3 Member Data Documentation

6.12.3.1 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.

6.12.3.2 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.

6.12.3.3 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 identify the color channels.

6.12.3.4 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 DOM \leftarrow HighResTimeStamp.

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

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

to: `frame.timestamp_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>

6.12.3.5 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

Subject to change

Index

addFace
 dfx::Frame, 40
addMarker
 dfx::Frame, 40
addToStudy
 dfx::DFXFactory, 31
addToStudyFromFile
 dfx::DFXFactory, 31
attributes
 dfx::Face, 37

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

cancelCollection
 dfx::Collector, 17
ChannelOrder
 dfx::VideoFrame, 53
checkConstraints
 dfx::Collector, 17
ChunkReady
 dfx, 11
Collecting
 dfx, 11
CollectorState
 dfx, 10
Completed
 dfx, 11
ConstraintResult
 dfx, 11
createCollector
 dfx::DFXFactory, 32
createFrame
 dfx::Collector, 17

decodeMeasurementResult
 dfx::Collector, 18
defineRegions
 dfx::Collector, 18
detected
 dfx::Face, 38
dfx, 9
 ChunkReady, 11
 Collecting, 11
 CollectorState, 10
 Completed, 11

 ConstraintResult, 11
 Error, 11
 Good, 11
 Waiting, 11
 Warn, 11
dfx::ChunkData, 13
 getChunkPayload, 13
dfx::ChunkPayload, 14
dfx::Collector, 15
 cancelCollection, 17
 checkConstraints, 17
 createFrame, 17
 decodeMeasurementResult, 18
 defineRegions, 18
 disableConstraint, 18
 enableConstraint, 19
 extractChannels, 19
 forceComplete, 20
 getAvailableConstraints, 20
 getChunkData, 20
 getChunkDurationSeconds, 20
 getCollectorState, 21
 getConstraintErrorMessage, 21
 getConstraintsConfig, 21
 getEnabledConstraints, 22
 getLastErrorMessage, 22
 getMode, 22
 getNumberChunks, 22
 getProperties, 23
 getProperty, 23
 getRequiredPosePointIDs, 23
 initializeModel, 23
 isChunkReady, 24
 numberFramesNeeded, 24
 prepareMeasurement, 24
 resetCollection, 25
 setChunkDurationSeconds, 25
 setCloudResultsFeedback, 25
 setConstraintsConfig, 26
 setFaceAttribute, 26, 27
 setNumberChunks, 27
 setProperty, 27
 setTargetFPS, 28
 startCollection, 28
 stopCollection, 29
dfx::DFXFactory, 29
 addToStudy, 31
 addToStudyFromFile, 31
 createCollector, 32

getLastErrorMessage, 32
 getMode, 32
 getProperty, 32
 getSdkID, 33
 getValidProperties, 33
 getVersion, 33
 initializeStudy, 34
 initializeStudyFromFile, 35
 removeProperty, 35
 setMode, 36
 setProperty, 36
 dfx::Error, 37
 dfx::Face, 37
 attributes, 37
 detected, 38
 faceRect, 38
 id, 38
 posePoints, 38
 poseValid, 38
 dfx::Frame, 39
 addFace, 40
 addMarker, 40
 getDesiredDeviceAttributes, 41
 getDesiredFaceAttributes, 41
 getDeviceAttribute, 41
 getFacelIdentifiers, 42
 getMarkers, 42
 getOpticalQualityMetrics, 42
 getOpticalQualityRating, 42
 getQualityMetrics, 43
 getRegionHistogram, 43
 getRegionIntProperty, 44
 getRegionNames, 44
 getRegionPolygon, 44
 getStarRating, 45
 getVideoFrame, 45
 setDeviceAttribute, 45
 dfx::MeasurementData, 46
 getData, 46
 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, 52
 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, 18
 enableConstraint
 dfx::Collector, 19
 Error
 dfx, 11
 estimated
 dfx::PosePoint, 51
 extractChannels
 dfx::Collector, 19
 faceRect
 dfx::Face, 38
 forceComplete
 dfx::Collector, 20
 getAvailableConstraints
 dfx::Collector, 20
 getChunkData
 dfx::Collector, 20
 getChunkDurationSeconds
 dfx::Collector, 20
 getChunkPayload
 dfx::ChunkData, 13
 getCollectorState
 dfx::Collector, 21
 getConstraintErrorMessage
 dfx::Collector, 21
 getConstraintsConfig
 dfx::Collector, 21
 getData
 dfx::MeasurementData, 46
 getDataProperty
 dfx::MeasurementData, 47
 getDataPropertyKeys
 dfx::MeasurementData, 47
 getDesiredDeviceAttributes
 dfx::Frame, 41
 getDesiredFaceAttributes
 dfx::Frame, 41
 getDeviceAttribute
 dfx::Frame, 41
 getEnabledConstraints
 dfx::Collector, 22
 getErrorCode
 dfx::MeasurementResult, 48

getFacelIdentifiers
 dfx::Frame, 42
getLastErrorMessage
 dfx::Collector, 22
 dfx::DFXFactory, 32
getMarkers
 dfx::Frame, 42
getMeasurementData
 dfx::MeasurementResult, 48
getMeasurementDataKeys
 dfx::MeasurementResult, 49
getMeasurementProperty
 dfx::MeasurementResult, 49
getMeasurementPropertyKeys
 dfx::MeasurementResult, 49
getMode
 dfx::Collector, 22
 dfx::DFXFactory, 32
getNumberChunks
 dfx::Collector, 22
getOpticalQualityMetrics
 dfx::Frame, 42
getOpticalQualityRating
 dfx::Frame, 42
getProperties
 dfx::Collector, 23
getProperty
 dfx::Collector, 23
 dfx::DFXFactory, 32
getQualityMetrics
 dfx::Frame, 43
getRegionHistogram
 dfx::Frame, 43
getRegionIntProperty
 dfx::Frame, 44
getRegionNames
 dfx::Frame, 44
getRegionPolygon
 dfx::Frame, 44
getRequiredPosePointIDs
 dfx::Collector, 23
getSdkID
 dfx::DFXFactory, 33
getStarRating
 dfx::Frame, 45
getValidProperties
 dfx::DFXFactory, 33
getVersion
 dfx::DFXFactory, 33
getVideoFrame
 dfx::Frame, 45
Good
 dfx, 11
Gray
 dfx::VideoFrame, 53
id
 dfx::Face, 38
image
 dfx::VideoFrame, 54
Infrared
 dfx::VideoFrame, 53
Infrared888
 dfx::VideoFrame, 53
initializeModel
 dfx::Collector, 23
initializeStudy
 dfx::DFXFactory, 34
initializeStudyFromFile
 dfx::DFXFactory, 35
isChunkReady
 dfx::Collector, 24
isValid
 dfx::MeasurementResult, 50
number
 dfx::VideoFrame, 54
numberFramesNeeded
 dfx::Collector, 24
order
 dfx::VideoFrame, 54
point
 dfx::PosePoint, 51
posePoints
 dfx::Face, 38
poseValid
 dfx::Face, 38
prepareMeasurement
 dfx::Collector, 24
quality
 dfx::PosePoint, 52
removeProperty
 dfx::DFXFactory, 35
resetCollection
 dfx::Collector, 25
RGB
 dfx::VideoFrame, 53
RGB_Infrared
 dfx::VideoFrame, 53
RGBA
 dfx::VideoFrame, 53
setChunkDurationSeconds
 dfx::Collector, 25
setCloudResultsFeedback
 dfx::Collector, 25
setConstraintsConfig
 dfx::Collector, 26
setDeviceAttribute
 dfx::Frame, 45
setFaceAttribute
 dfx::Collector, 26, 27
setMode
 dfx::DFXFactory, 36
setNumberChunks

dfx::Collector, 27
setProperty
 dfx::Collector, 27
 dfx::DFXFactory, 36
setTargetFPS
 dfx::Collector, 28
startCollection
 dfx::Collector, 28
stopCollection
 dfx::Collector, 29

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

valid
 dfx::PosePoint, 52

Waiting
 dfx, 11
Warn
 dfx, 11