



Class List

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

[nite::Array< T >](#)[nite::BoundingBox](#)[nite::GestureData](#)[nite::HandData](#)[nite::HandTracker](#)[nite::HandTrackerFrameRef](#)[nite::HandTracker::NewFrameListener](#)[nite::UserTracker::NewFrameListener](#)[nite::NiTE](#)[NiteBoundingBox](#)[NitePlane](#)[NitePoint3f](#)[NiteQuaternion](#)[nite::Plane](#)[nite::Point3f](#)[nite::PoseData](#)[nite::Quaternion](#)[nite::Skeleton](#)[nite::SkeletonJoint](#)[nite::UserData](#)

nite::UserMap

nite::UserTracker

nite::UserTrackerFrameRef

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [**doxygen**](#) 1.7.5.1



nite::Array< T > Class Template Reference

#include <[NiTE.h](#)>

List of all members.

Public Member Functions

```
    Array ()  
    int getSize () const  
    bool isEmpty () const  
    const T & operator[] (int index) const  
    void setData (int size, T *data)
```

Detailed Description

```
template<class T>
class nite::Array< T >
```

Provides a simple array class used throughout the API. Wraps a primitive array of objects, holding the elements and their count.

Constructor & Destructor Documentation

```
template<class T>
nite::Array< T >::Array( ) [inline]
```

Default constructor. Creates an empty **Array** and sets the element count to zero.

Member Function Documentation

```
template<class T>
int nite::Array< T >::getSize( ) const [inline]
```

Getter function for the **Array** size.

Returns:

Current number of elements in the **Array**.

```
template<class T>
bool nite::Array< T >::isEmpty( ) const [inline]
```

Check if there are any elements in the **Array**

Returns:

true if there are elements in the **Array**, false otherwise.

```
template<class T>
const T& nite::Array< T >::operator[]( int index ) const [inline]
```

Implements the array indexing operator for the **Array** class

```
template<class T>
void nite::Array< T >::setData( int size,
                                 T * data
                               ) [inline]
```

Setter function for data. Causes this **Array** to wrap an existing primitive array of specified type.

Parameters:

[in] **T** Type of objects the **Array** will contain

[in] **data** Pointer to first object in list
[in] **size** Number of objects in list

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

- [NiTE.h](#)

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



NiTE 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Hierarchy](#)[Class Members](#)[nite](#) › [BoundingBox](#) ›

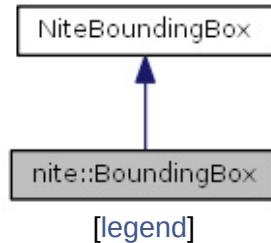
Public Member Functions

nite::BoundingBox

Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::BoundingBox:



List of all members.

Public Member Functions

`BoundingBox ()`

`BoundingBox (const Point3f &min, const Point3f &max)`

Detailed Description

Represents a box in three dimensional space. The box is represented as two points, containing respectively its minimum and maximum x, y and z coordinate values.

Constructor & Destructor Documentation

nite::BoundingBox::BoundingBox () [inline]

Default constructor. Does nothing.

nite::BoundingBox::BoundingBox (const Point3f & **min,
const Point3f & **max**
)** [inline]

Constructor. Creates a bounding box from two points in space, which represent that minimum and maximum value of all of its coordinates.

Parameters:

- [in] **min** A point containing the minimum x, y and z values of the bounding box.
- [in] **max** A point containing the maximum x, y and z values of the bounding box.

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

- **NiTE.h**

Main Page

Namespaces

Classes

Files

Class List

Class Index

Class Hierarchy

Class Members

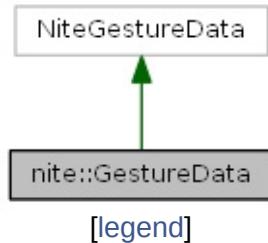
nite > GestureData >

Public Member Functions

nite::GestureData Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::GestureData:



List of all members.

Public Member Functions

```
const Point3f & getCurrentPosition () const  
GestureType getType () const  
bool isComplete () const  
bool isInProgress () const
```

Detailed Description

This class stores data on a gesture that is being detected. "Gestures" in this context indicate gestures detected directly from the depthmap, so there are no handpoint ID's at this stage. Objects of this class store the data for a specific instance of a specific gesture.

The type, position in space, and completion status of the gesture can all be tracked.

All positions are given in "real world" coordinates. See the OpenNI 2.0 documentation for more information on the coordinate systems used by OpenNI/NiTE.

See also:

[GestureType](#) enumeration for a list of available hand gestures.
[HandTracker](#) for more information on using gestures.

Member Function Documentation

`const Point3f& nite::GestureData::getCurrentPosition() const [in]`

This function provides the position in space of the projected gesture, using "real world" coordinates.

Returns:

The position of the gesture being detected.

`GestureType nite::GestureData::getType() const [inline]`

This function is used to find out what type of gesture that this object is storing detection data for.

Returns:

the type of gesture.

See also:

[GestureType](#) for a list of valid gestures.

`bool nite::GestureData::isComplete() const [inline]`

This function indicates whether the complete gesture has been detected yet.

Returns:

True if the gesture is complete, false otherwise.

See also:

[isInProgress\(\)](#) to detect gestures that are only partially complete.

bool nite::GestureData::isInProgress() const [inline]

This function indicates whether a non-completed gesture has been detected. This can be useful for providing user feedback during the gesture.

Returns:

True if a gesture is currently being detected. False if detection is complete or has not yet started.

See also:

[isComplete\(\)](#) to detect only the completed gesture rather than a gesture in progress.

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

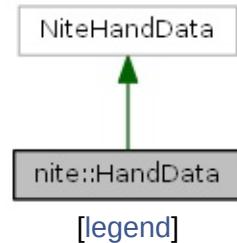
- [NiTE.h](#)



nite::HandData Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::HandData:



List of all members.

Public Member Functions

```
    HandId getId () const
const Point3f & getPosition () const
    bool isLost () const
    bool isNew () const
    bool isTouchingFov () const
    bool isTracking () const
```

Detailed Description

This class stores data on a single hand during a single frame of hand detection by the .

It can be used to find out where the hand is in space, the hand's ID, and the tracking status.

All positions are given in "real world" coordinates. See the OpenNI 2.0 documentation for more information on the coordinate systems used by OpenNI/NiTE.

See also:

[HandTracker](#) for more information on hand tracking.

Member Function Documentation

HandId `nite::HandData::getId() const [inline]`

Get the ID of the hand referred to by this data.

Returns:

HandId of this hand.

const Point3f& `nite::HandData::getPosition() const [inline]`

Get the position in space of the hand during this frame.

Returns:

Position of the hand.

bool `nite::HandData::isLost() const [inline]`

Indicates whether the hand tracking algorithm has lost track of the hand. This can happen, for example, if the hand leaves the field of view of the sensor or becomes occluded.

Returns:

True if this hand is no longer being tracked.

bool `nite::HandData::isNew() const [inline]`

Indicates if the hand referred to by this object is newly tracked.

Returns:

True if this is the first frame that this hand has been detected in, false otherwise.

bool nite::HandData::isTouchingFov() const [inline]

Indicates whether the hand is at the edge of the sensor field of view.

Returns:

True if the hand is at the edge of the field of view, false otherwise.

bool nite::HandData::isTracking() const [inline]

Indicates whether this hand is being actively tracked.

Returns:

True if hand is being successfully tracked, false otherwise.

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

- [NiTE.h](#)



nite::HandTracker

Class Reference

```
#include <NiTE.h>
```

List of all members.

Classes

class **NewFrameListener**

Public Member Functions

	HandTracker ()
	~HandTracker ()
void	addNewFrameListener (NewFrameListener *pListener)
Status	convertDepthCoordinatesToHand (int x, int y, int z, float *pOutX, float *pOutY) const
Status	convertHandCoordinatesToDepth (float x, float y, float z, float *pOutX, float *pOutY) const
Status	create (openni::Device *pDevice=NULL)
void	destroy ()
float	getSmoothingFactor () const
bool	isValid () const
Status	readFrame (HandTrackerFrameRef *pFrame)
void	removeNewFrameListener (NewFrameListener *pListener)
Status	setSmoothingFactor (float factor)
Status	startGestureDetection (GestureType type)
Status	startHandTracking (const Point3f &position, HandId *pNewHandId)
void	stopGestureDetection (GestureType type)
void	stopHandTracking (HandId id)

Detailed Description

This is the main object of the Hand Tracker algorithm. It (along with [UserTracker](#)) is one of two main classes in [NiTE](#). All [NiTE](#) algorithms are accessible through one of these two classes.

[HandTracker](#) provides access to all algorithms relates to tracking individual hands, as well as detecting gestures in the depthmap.

The core of the hand tracking is an algorithm that finds human hands in each from of the depthmap, and reports the position of those hands in space. This can be used for simple detection of higher level gestures and implementation of gesture based user interfaces. Unlike full body tracking algorithms, handpoint based tracking works on users that are sitting and does not require a full body be visible.

Gesture tracking is generally used to initiate hand tracking. It allows detection of gestures in the raw depth map, without requiring hand points (in contrast to higher-level gestures that might be used to implement a UI using handpoints). These gestures can be located in space to provide a hint to the hand tracking algorithm on where to start tracking.

The output of the [HandTracker](#) occurs one frame at a time. For each input depth frame, a hand tracking frame is output with hand positions, gesture positions, etc. A listener class is provided that allows for event driven reaction to each new frame as it arrives.

Note that creating a [HandTracker](#) requires a valid OpenNI 2.0 Device to be initialized in order to provide depth information. See the OpenNI 2.0 documentation for information on connecting a device and starting the stream of depth maps that will drive this algorithm.

See also:

[UserTracker](#) If you want to track full body motion, skeletons, find a floor plane, or detect poses.

[NiTE](#) For a list of static functions that must be used before using

HandTracker

Constructor & Destructor Documentation

`nite::HandTracker::HandTracker() [inline]`

Default constructor. Creates an empty [HandTracker](#) with a NULL handle. This object will not be useful until the [create\(\)](#) function is called.

See also:

[HandTracker::create\(\)](#) for a function to [create](#) and activate the algorithm.

[HandTracker::isValid\(\)](#) to determine whether [create\(\)](#) has already been called.

`nite::HandTracker::~HandTracker() [inline]`

Destructor. Automatically calls the provided [destroy\(\)](#) function.

Member Function Documentation

void nite::HandTracker::addNewFrameListener ([NewFrameListene](#)

Adds a NewFrameListner object to this [HandTracker](#) so that it will respond when a new frame is generated.

Parameters:

[in] **pListener** Pointer to a listener to add.

See also:

[HandTracker::NewFrameListener](#) for more information on using event based interaction with [HandTracker](#)

Status nite::HandTracker::convertDepthCoordinatesToHand (int int int float float)

In general, two coordinate systems are used in OpenNI 2.0. These conventions are also followed in [NiTE](#) 2.0.

Hand point and gesture positions are provided in "Real World" coordinates, while the native coordinate system of depth maps is the "projective" system. In short, "Real World" coordinates locate objects using a Cartesian coordinate system with the origin at the sensor. "Projective" coordinates measure straight line distance from the sensor, and indicate x/y coordinates using pixels in the image (which is mathematically equivalent to specifying angles). See the OpenNI 2.0 documentation online for more information.

This function allows you to convert the native depth map coordinates

to the system used by the hand points. This might be useful for performing certain types of measurements (eg distance between a hand and an object identified only in the depth map).

Note that no output is given for the Z coordinate. Z coordinates remain the same when performing the conversion. An input value is still required for Z, since this can affect the x/y output.

Parameters:

- [in] **x** The input X coordinate using the "projective" coordinate system.
- [in] **y** The input Y coordinate using the "projective" coordinate system.
- [in] **z** The input Z coordinate using the "projective" coordinate system.
- [out] **pOutX** Pointer to a location to store the output X coordinate in the "real world" system.
- [out] **pOutY** Pointer to a location to store the output Y coordinate in the "real world" system.

Returns:

Status indicating success or failure of this operation. This is needed because the ability to convert between coordinate systems requires a properly initialized Device from OpenNI 2.0.

```
Status nite::HandTracker::convertHandCoordinatesToDepth ( float  
                           float  
                           float  
                           float  
                           float  
                           )
```

In general, two coordinate systems are used in OpenNI 2.0. These conventions are also followed in **NiTE** 2.0.

Hand point and gesture positions are provided in "Real World" coordinates, while the native coordinate system of depth maps is the "projective" system. In short, "Real World" coordinates locate objects using a Cartesian coordinate system with the origin at the sensor. "Projective" coordinates measure straight line distance from the sensor (perpendicular to the sensor face), and indicate x/y coordinates using pixels in the image (which is mathematically equivalent to specifying angles). See the OpenNI 2.0 documentation online for more information.

Note that no output is given for the Z coordinate. Z coordinates remain the same when performing the conversion. An input value is still required for Z, since this can affect the x/y output.

This function allows you to convert the coordinates of a hand point or gesture to the native coordinates of a depth map. This is useful if you need to find the hand position on the raw depth map.

Parameters:

- [in] **x** The input X coordinate using the "real world" coordinate system.
- [in] **y** The input Y coordinate using the "real world" coordinate system.
- [in] **z** The input Z coordinate using the "real world" coordinate system.
- [out] **pOutX** Pointer to a location to store the output X coordinate in the "projective" system.
- [out] **pOutY** Pointer to a location to store the output Y coordinate in the "projective" system.

Returns:

Status indicating success or failure of this operation. This is needed because the ability to convert between coordinate systems requires a properly initialized Device from OpenNI 2.0.

Status nite::HandTracker::create (openni::Device * pDevice = NULL

Creates and initializes an empty **HandTracker**. This function should be the first one called when a new **HandTracker** object is constructed.

An OpenNI device with depth capabilities is required for this algorithm to work. See the OpenNI 2.0 documentation for more information about using an OpenNI 2.0 compliant hardware device and creating a Device object.

Parameters:

[in] **pDevice** A pointer to an initialized OpenNI 2.0 Device object that provides depth streams.

Returns:

A status code to indicate success/failure. Since this relies on an external hardware device, it is important for applications to check this value.

See also:

Status enumeration for a list of all possible status values generated by this call.

void nite::HandTracker::destroy() [inline]

Shuts down the hand tracker and releases all resources used by it.

This is the opposite of **create()**. This function is called automatically by the destructor in the current implementation, but it is good practice to run it manually when the algorithm is no longer required. Running this function more than once is safe -- it simply exits if called on a non-valid **HandTracker**.

float nite::HandTracker::getSmoothingFactor() const [inline]

Queries the current hand smoothing factor.

Returns:

Current hand smoothing factor.

See also:

[setSmoothingFactor](#) for more information on the smoothing factor, and the means to change it.

`bool nite::HandTracker::isValid () const [inline]`

Indicates whether the [HandTracker](#) is valid.

When a new [HandTracker](#) is first constructed, this function will indicate that it is invalid (ie return False). Once the [create\(\)](#) function has been successfully called, then this function will return True. If the [destroy\(\)](#) function is called, this function will again indicate invalid.

It is safe to run [create\(\)](#) and [destroy\(\)](#) without calling this function -- both of those functions already check this value and return without doing anything if no action is required.

Returns:

True if the [HandTracker](#) object is correctly initialized, False otherwise.

See also:

[create\(\)](#) function -- causes the [UserTracker](#) to become initialized.

[destroy\(\)](#) function -- causes the [UserTracker](#) to become uninitialized.

`Status nite::HandTracker::readFrame (HandTrackerFrameRef * pF)`

Gets the next snapshot of the algorithm. This causes all data to be generated for the next frame of the algorithm -- algorithm frames correspond to the input depth frames used to generate them.

Parameters:

pFrame [out] A pointer that will be set to point to the next frame of data.

Returns:

Status code indicating whether this operation was successful.

void nite::HandTracker::removeNewFrameListener ([NewFrameList](#)

Removes a [NewFrameListener](#) object from this HandTracker's list of listeners. The listener will no longer respond when a new frame is generated.

Parameters:

[in] **pListener** Pointer to a listener to remove.

See also:

[HandTracker::NewFrameListener](#) for more information on using event based interaction with [HandTracker](#).

Status nite::HandTracker::setSmoothingFactor (float **factor) [inli**

Control the smoothing factor of the hand points. Factor should be between 0 (no smoothing at all) and 1 (no movement at all).

Experimenting with this factor should allow you to fine tune the hand tracking performance. Higher values will produce smoother movement of the handpoints, but may make the handpoints feel less responsive to the user.

Parameters:

[in] **factor** The smoothing factor.

Returns:

Status code indicating success or failure of this operation.

Status nite::HandTracker::startGestureDetection (GestureType type)

Start detecting a specific gesture. This function will cause the algorithm to start scanning the entire field of view for any hand that appears to be performing the gesture specified. Intermediate progress is available to aid in providing feedback to the user.

Gestures are detected from the raw depth map. They don't depend on hand points. They are most useful for determining where a hand is in space to start hand tracking. Unlike handpoints, they do not follow a specific hand, so they will react to a hand anywhere in the room.

If you want to detect user gestures for input purposes, it is often better to use a single "focus" gesture to start hand tracking, and then detect other gestures from the handpoints. This enables an application to focus on a single user, even in a crowded room.

Hand points can also be more computationally efficient. The gesture tracking algorithm for any given gesture uses about as much CPU bandwidth as the hand tracker. Adding more gestures or also running the hand tracker increases CPU consumption linearly. Finding gestures from hand points, on the other hand, can be done for negligible CPU cost once the handpoint algorithm has run. This means that user interface complexity will scale better with CPU complexity.

Parameters:

[in] @ref GestureType you wish to detect

Returns:

Status indicating success or failure of this operation.

See also:

[UserTracker](#) if you want to do full body poses instead of hand-only gestures

[GestureType](#) enumeration for list of available gestures.

`stopGestureDetection` to stop detection once it has started.

**Status nite::HandTracker::startHandTracking (const Point3f & pos
HandId * HandId * pNe
) [in]**

Starts tracking a hand at a specific point in space. Use of this function assumes that there actually is a hand in the location given. In general, the hand algorithm is much better at tracking a specific hand as it moves around than it is at finding the hand in the first place.

This function is typically used in conjunction with gesture detection. The position in space of the gesture is used to initiate hand tracking. It is also possible to start hand tracking without a gesture if your application will constrain users to place their hands in a certain known point in space. A final possibility is for applications or third party middleware to implement their own hand 'finding' algorithm either in depth or from some other information source, and using that data to initialize the hand tracker.

The position in space of the hand point is specified in "real world" coordinates. See OpenNI 2.0 documentation for more information on coordinate systems.

Parameters:

- [in] **Point** where hand is known/suspected to exist.
- [in] **ID** to assign a hand once tracking starts. This will be used to refer to the hand later.

Returns:

Status code indicating success or failure of this operation.

void nite::HandTracker::stopGestureDetection (GestureType type)

Stop detecting a specific gesture. This disables detection of the

specified gesture. Doing this when that gesture is no longer required prevents false detection and saves CPU bandwidth.

Parameters:

[in] **@ref** GestureType you would like to stop detecting.

void nite::HandTracker::stopHandTracking (HandId id) [inline]

Commands the algorithm to stop tracking a specific hand. Note that the algorithm may be tracking more than one hand. This function only halts tracking on the single hand specified.

Parameters:

[in] **id** The **HandId** of the hand to quit tracking.

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

- **NiTE.h**



nite::HandTrackerFrameRef Class Reference

#include <[NiTE.h](#)>

List of all members.

Public Member Functions

		HandTrackerFrameRef ()
		HandTrackerFrameRef (const HandTrackerFrameRef &other)
		~HandTrackerFrameRef ()
openni::VideoFrameRef		getDepthFrame () const
	int	getFrameIndex () const
const Array< GestureData > &		getGestures () const
const Array< HandData > &		getHands () const
	uint64_t	getTimestamp () const
	bool	isValid () const
HandTrackerFrameRef &		operator= (const HandTrackerFrameRef &other)
	void	release ()

Friends

```
class HandTracker
```

Detailed Description

Snapshot of the Hand Tracker algorithm. It contains all output from a single frame of the Hand Tracking algorithm, which corresponds to a single input depth frame.

It holds all the hands identified at this time, as well as the detected gestures.

See also:

[HandTracker](#) for how this data is created

Constructor & Destructor Documentation

`nite::HandTrackerFrameRef::HandTrackerFrameRef() [inline]`

Default constructor. Creates a new empty `HandTrackerFrameRef`. This object will not be valid or useful until it has been initialized with data from a specific frame.

`nite::HandTrackerFrameRef::~HandTrackerFrameRef() [inline]`

Destructor. Calls the provided `release()` function.

`nite::HandTrackerFrameRef::HandTrackerFrameRef(const HandTr`

Copy constructor. Creates a `HandTrackerFrameRef` that is identical to an existing one.

Parameters:

[in] `@ref HandTrackerFrameRef` to be copied.

Member Function Documentation

`openni::VideoFrameRef nite::HandTrackerFrameRef::getDepthFrame()`

Provides the raw depth frame used to generate this Hand Tracker frame. See the OpenNI 2.0 documentation for more information on manipulating this data.

Returns:

`VideoFrameRef` used to create this object.

`int nite::HandTrackerFrameRef::getFrameIndex() const [inline]`

Gets the frame index of the raw depth frame that generated this frame. Frame indexes are consecutive integer numbers assigned to frames by OpenNI. See the OpenNI documentation for more information.

Returns:

`FrameIndex` of the depth frame used to generate this User Tracker frame.

`const Array<GestureData>& nite::HandTrackerFrameRef::getGestures()`

This function provides a list of all gestures being actively detected in this frame.

Returns:

`Array` of `GestureData` objects, one for each gesture being detected.

`const Array<HandData>& nite::HandTrackerFrameRef::getHands()`

This function provides a list of all hands being tracked in this frame.

Returns:

Array of **HandData** objects, one for each hand being tracked.

uint64_t nite::HandTrackerFrameRef::getTimestamp() const [inline]

Get the timestamp in which this frame was processed. See the OpenNI 2.0 documentation for more information on how OpenNI assigns time stamps to depth frames.

Returns:

Timestamp in microseconds

bool nite::HandTrackerFrameRef::isValid() const [inline]

Indicates whether this object has been loaded with valid data.

Returns:

True if this object contains valid data, false otherwise.

HandTrackerFrameRef& nite::HandTrackerFrameRef::operator=(const HandTrackerFrameRef &other) [inline]

Operator. Implements the assignment operator for this class.

Parameters:

[in] **other HandTrackerFrameRef** to be assigned.

void nite::HandTrackerFrameRef::release() [inline]

Properly releases all resources used by this object. This function is automatically called by the destructor, but it is a good practice to

release the resources as soon as they are no longer required. It is safe to run this function multiple times on the same object.

Friends And Related Function Documentation

friend class HandTracker [friend]

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

- [NiTE.h](#)

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



nite::HandTracker::NewFrameListener Class Reference

#include <**NiTE.h**>

List of all members.

Public Member Functions

NewFrameListener ()

virtual void **onNewFrame (HandTracker &)=0**

Friends

```
class HandTracker
```

Detailed Description

This is a listener class that is used to react to events generated by the [HandTracker](#) class.

To use this class, you must derive a class from it that implements the [onNewFrame\(\)](#) function. This is the function that will be called when an event is generated.

Create a new instance of your derived class. Then, use the [HandTracker::addNewFrameListener\(\)](#) function to add the listener to the [HandTracker](#). When that [HandTracker](#) generates an onNewFrame event, \ the specified callback function will be called.

The onNewFrame event is currently the only event type that this listener is designed to work with.

See also:

[HandTracker](#) for the source of this listener's events.

Constructor & Destructor Documentation

nite::HandTracker::NewFrameListener::NewFrameListener() [inli

Default Constructor. Creates a new object of this type and configures it to correctly receive events.

Member Function Documentation

virtual void nite::HandTracker::NewFrameListener::onNewFrame (

This is the callback function for the event. It should be implemented in a class derived from [NewFrameListener](#). This function will automatically be called when the OnNewFrame event is triggered.

Parameters:

[in] A reference to the [HandTracker](#) that triggered the event is provided.

Friends And Related Function Documentation

friend class HandTracker [friend]

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

- [NiTE.h](#)

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



nite::UserTracker::NewFrameListener Class Reference

#include <**NiTE.h**>

List of all members.

Public Member Functions

NewFrameListener ()

virtual void **onNewFrame (UserTracker &)=0**

Friends

```
class UserTracker
```

Detailed Description

This is a listener class that is used to react to events generated by the [UserTracker](#) class.

To use this class, you must derive a class from it that implements the [onNewFrame\(\)](#) function. This is the function that will be called when an event is generated.

Create a new instance of your derived class. Then, use the [UserTracker::addNewFrameListener\(\)](#) function to add the listener to the [UserTracker](#). When that [UserTracker](#) generates an onNewFrame event, \ the specified callback function will be called.

The onNewFrame event is currently the only event type that this listener is designed to work with.

See also:

[UserTracker](#) for the source of this listener's events.

Constructor & Destructor Documentation

nite::UserTracker::NewFrameListener::NewFrameListener() [inlin

Default Constructor. Creates a new object of this type and configures it to correctly receive events.

Member Function Documentation

virtual void nite::UserTracker::NewFrameListener::onNewFrame ([L](#))

This is the callback function for the event. It should be implemented in a class derived from [NewFrameListener](#). This function will automatically be called when the OnNewFrame event is triggered.

Parameters:

[in] A reference to the [UserTracker](#) that triggered the event is provided.

Friends And Related Function Documentation

friend class UserTracker [friend]

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

- [NiTE.h](#)

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



NiTE 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Hierarchy](#)[Class Members](#)[nite](#)[NiTE](#)

Static Public Member Functions

nite::NiTE Class Reference

```
#include <NiTE.h>
```

List of all members.

Static Public Member Functions

```
static Version getVersion()  
static Status initialize()  
static void shutdown()
```

Detailed Description

The **NiTE** class is a static entry point to the library.

All applications that expect to use the **NiTE** middleware should run the **NiTE.initialize()** function before using any other API calls. Before the application exits, the application should run the **NiTE.shutdown()** function to properly close the API.

Member Function Documentation

static Version nite::NiTE::getVersion() [inline, static]

This function allows you to obtain the current version of the **NiTE** installation. This is useful for verifying application compatibility with the currently used version of **NiTE**, and for reporting this information to the user for troubleshooting purposes. In general, there should be minimal version conflict problems, as **NiTE** and OpenNI applications each contain their own copy of the API runtime files.

Returns:

The version of the current **NiTE** installation.

static Status nite::NiTE::initialize() [inline, static]

Initializes the **NiTE** API. This function should be called before using any other API functionality.

Returns:

A status code of type **Status** that indicates success/failure of the initialization.

static void nite::NiTE::shutdown() [inline, static]

Shuts down the API, properly releasing all resources used by **NiTE**. This should be called by all **NiTE** applications before they close.

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

- **NiTE.h**

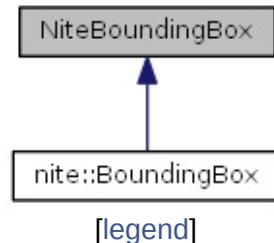
Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



NiteBoundingBox Struct Reference

```
#include <NiteCTypes.h>
```

Inheritance diagram for NiteBoundingBox:



List of all members.

Public Attributes

NitePoint3f max

NitePoint3f min

Detailed Description

3D Box

Member Data Documentation

NitePoint3f NiteBoundingBox::max

NitePoint3f NiteBoundingBox::min

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

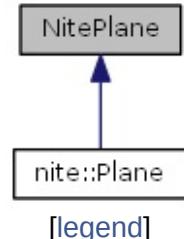
- **NiteCTypes.h**

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1

NitePlane Struct Reference

```
#include <NiteCTypes.h>
```

Inheritance diagram for NitePlane:



List of all members.

Public Attributes

`NitePoint3f normal`

`NitePoint3f point`

Detailed Description

3D Plane

Member Data Documentation

NitePoint3f NitePlane::normal

NitePoint3f NitePlane::point

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

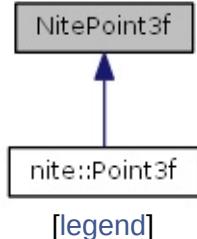
- **NiteCTypes.h**

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1

NitePoint3f Struct Reference

```
#include <NiteCTypes.h>
```

Inheritance diagram for NitePoint3f:



List of all members.

Public Attributes

float `x`

float `y`

float `z`

Detailed Description

3D Point

Member Data Documentation

float NitePoint3f::x

float NitePoint3f::y

float NitePoint3f::z

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

- **NiteCTypes.h**

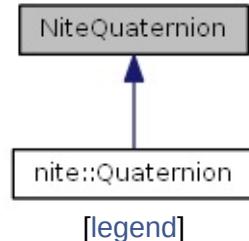
Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



NiteQuaternion Struct Reference

```
#include <NiteCTypes.h>
```

Inheritance diagram for NiteQuaternion:



List of all members.

Public Attributes

float `w`

float `x`

float `y`

float `z`

Detailed Description

Quaternion

Member Data Documentation

float NiteQuaternion::w

float NiteQuaternion::x

float NiteQuaternion::y

float NiteQuaternion::z

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

- [NiteCTypes.h](#)

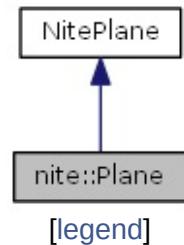
Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



nite::Plane Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::Plane:



List of all members.

Public Member Functions

`Plane ()`

`Plane (const Point3f &point, const Point3f &normal)`

Detailed Description

Enapsulates a geometrical plane. The plane is defined by a single point on that plane, and a normal vector to the plane. The normal vector is represented as a [Point3f](#).

Constructor & Destructor Documentation

`nite::Plane::Plane()` [inline]

Default constructor. Creates a plane that passes through the origin, and has a normal vector set to (0,0,0). Note that the default normal vector will make the plane generated by this function invalid for many mathematical operations, as it is a degenerate vector and has no direction.

`nite::Plane::Plane(const Point3f & point,
const Point3f & normal
)` [inline]

Constructor. Creates a new point, given a single point on that plane and a normal vector.

Parameters:

- [in] **point** Any point on the plane
- [in] **normal** The normal vector of the plane, represented as a **Point3f**

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

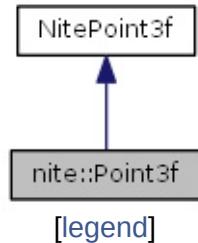
- **NiTE.h**



nite::Point3f Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::Point3f:



List of all members.

Public Member Functions

Point3f ()
Point3f (float *x*, float *y*, float *z*)
Point3f (const Point3f &other)
bool **operator!=** (const **Point3f** &other) const
Point3f & operator= (const **Point3f** &other)
bool **operator==** (const **Point3f** &other) const
void **set** (float *x*, float *y*, float *z*)

Detailed Description

Encapsulates a single point in 3D space, storing the x/y/z coordinates as floating point numbers.

Can also be used to represent a three dimensional vector.

Constructor & Destructor Documentation

nite::Point3f::Point3f() [inline]

Default Constructor, creates a point with all three coordinate values set to 0.0

**nite::Point3f::Point3f(float *x*,
 float *y*,
 float *z*
) [inline]**

Constructor. Creates a point from three given coordinates.

Parameters:

- [in] *x* X coordinate of point to be created
- [in] *y* Y coordinate of point to be created
- [in] *z* Z coordinate of point to be created

nite::Point3f::Point3f(const Point3f & *other*) [inline]

Copy Constructor. Creates a new point that has the same coordinates as an existing point.

Parameters:

- [in] *other* An existing point to be copied.

Member Function Documentation

bool nite::Point3f::operator!= (const Point3f & other) const [inline]

Inequality operator. Tests two points to see if they are different.

Parameters:

[in] **other** One of two points to test

Returns:

True if either x, y, or z coordinate of first point differs from the matching coordinate of the second point.

Point3f& nite::Point3f::operator= (const Point3f & other) [inline]

Assignment operator. Sets all coordinates of the point on the left to be equal to the coordinate values of the point on the right.

Parameters:

[in] **other** Point to copy coordinates from

Returns:

Reference to a point containing a copy of the input points coordinates.

bool nite::Point3f::operator== (const Point3f & other) const [inline]

Equality operator. Tests two points to determine whether they have matching coordinate values.

Parameters:

[in] **other** One of the points to be tested against

Returns:

True if all three coordinate values of left point match all three coordinate values of right point.

```
void nite::Point3f::set ( float x,  
                         float y,  
                         float z  
                     ) [inline]
```

Setter function for all coordinates of the point.

Parameters:

- [in] **x** Desired new X coordinate of the point.
- [in] **y** Desired new Y coordinate of the point.
- [in] **z** Desired new Z coordinate of the point.

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

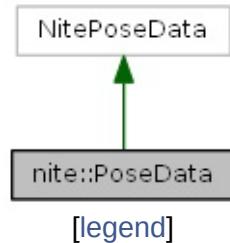
- [NiTE.h](#)



nite::PoseData Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::PoseData:



List of all members.

Public Member Functions

```
PoseType getType () const
    bool isEntered () const
    bool isExited () const
    bool isHeld () const
```

Detailed Description

This class wraps the output from the [NiTE](#) Pose Detection algorithms.

[NiTE](#) provides the ability for applications to detect whether a user is in a specific pose. This is most commonly used for skeleton calibration. The algorithm passes through a number of different states during the process of Pose Detection.

For each user being analyzed, this class stores the pose being detected and the state of that detection.

Member Function Documentation

PoseType `nite::PoseData::getType() const [inline]`

Get the type of this pose.

Returns:

The type of pose being detected, formatted as a **PoseType**.

bool `nite::PoseData::isEntered() const [inline]`

Checks whether this is the first frame in which the user has assumed the pose indicated by **PoseData::getType()**

Returns:

True if the user is in the pose indicated AND the user was not in that pose on the previous frame. Returns false otherwise.

bool `nite::PoseData::isExited() const [inline]`

Check if the user has exited the pose in this frame.

Returns:

True if the user is NOT in the pose indicated, and was in the previous frame. Returns false otherwise.

bool `nite::PoseData::isHeld() const [inline]`

Checks if the user is currently in the pose indicated by the **PoseData::getType()** function.

Returns:

True if the user is in the pose indicated, False otherwise.

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

- [NiTE.h](#)
-

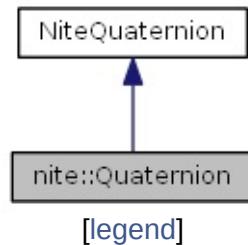
Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



nite::Quaternion Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::Quaternion:



List of all members.

Public Member Functions

[**Quaternion \(\)**](#)

[**Quaternion \(float w, float x, float y, float z\)**](#)

Detailed Description

Represents a **Quaternion**. The **Quaternion** is stored as four floating point numbers. (The quaternions are a number system that extends the complex number system from two dimensions to four.)

Constructor & Destructor Documentation

nite::Quaternion::Quaternion () [inline]

Default Constructor. Creates a new **Quaternion** with the value of all four components set to zero.

**nite::Quaternion::Quaternion (float w,
float x,
float y,
float z
)** [inline]

Constructor. Creates a new **Quaternion** with each component specified.

Parameters:

- [in] **w** Desired first quaternion component value.
- [in] **x** Desired second quaternion component value.
- [in] **y** Desired third quaternion component value.
- [in] **z** Desired fourth quaternion component value.

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

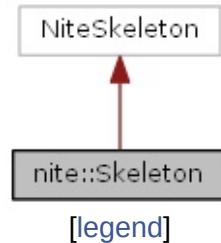
- [NiTE.h](#)



nite::Skeleton Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::Skeleton:



List of all members.

Public Member Functions

```
const SkeletonJoint & getJoint (JointType type) const  
SkeletonState getState () const
```

Detailed Description

This is the main class for the skeleton algorithm.

The purpose of the skeleton algorithm is to analyze a user outline supplied by the User Tracker algorithm, and to locate the position of that user's joints in space (eg knees, elbows, head, etc). The orientation of the user's joints is also calculated. Where joints are not visible, the algorithm will make a best guess about the joint. For all data calculated, confidence values are also created to help an application understand if the algorithm is sure about the data, or if it is "guessing".

Nite offers two types of skeletons algorithms. One requires the user to assume a specific "calibration pose" before calibration can be attempted. The other does not require a specific pose from the user, but will require a few seconds to settle down as it learns a user's body shape. The algorithm used sometimes depends on the platform being used -- for example, the ARM version of Nite currently only offers the version that requires a calibration pose.

All skeleton joint positions are given in "real world" data coordinates, using the same conventions as OpenNI 2.0. See the online documentation for OpenNI 2.0 for more information on the coordinate systems used with the skeleton.

See also:

[SkeletonJoint](#) for the output data format of the skeleton.

[JointType](#) enumeration for a list of all joints detected by the skeleton.

[UserTracker](#) to understand where the input data to the skeleton comes from.

[SkeletonState](#) enumeration for a list of possible skeleton states.

Member Function Documentation

`const SkeletonJoint& nite::Skeleton::getJoint(JointType type) co`

Get a specific joint of the skeleton. The object returned by this function can be queried to determine position, orientation, and confidence values for the joint.

Parameters:

[in] **type** The type of the joint to retrieve from the skeleton.

Returns:

Pointer to a **SkeletonJoint** containing the requested data.

See also:

JointType for a list of possible joints to request from the skeleton.

SkeletonState `nite::Skeleton::getState() const [inline]`

Get the state of the skeleton. This is useful for determining whether an active skeleton exists, and whether it is calibrating. Some of the states are also error conditions that attempt to provide a hint about why the skeleton is not calibrating. This can be useful for providing feedback to a user about where they need to stand, assuming a calibration pose, etc.

Returns:

Current state of the skeleton.

See also:

SkeletonState enumeration for a list of possible states

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

- [NiTE.h](#)
-

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



NiTE 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Hierarchy](#)[Class Members](#)[nite](#) › [SkeletonJoint](#) ›

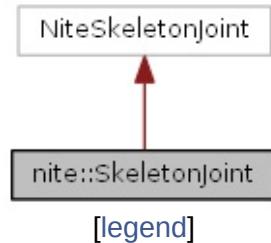
Public Member Functions

nite::SkeletonJoint

Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::SkeletonJoint:



List of all members.

Public Member Functions

```
const Quaternion & getOrientation () const  
    float getOrientationConfidence () const  
const Point3f & getPosition () const  
    float getPositionConfidence () const  
JointType getType () const
```

Detailed Description

This class provides a data structure to store the output of the [Skeleton](#) algorithm.

The basic purpose of the skeleton algorithm is to locate each of a user's joints, and to estimate the position and orientation of each joint in space. The output from the [Skeleton](#) algorithm is stored as a series of [SkeletonJoint](#) objects.

See also:

[nite::Skeleton](#) for more information about the [Skeleton](#) algorithm.

Member Function Documentation

`const Quaternion& nite::SkeletonJoint::getOrientation() const [inline]`

Get the current orientation of the joint represented by this object. This orientation will be represented by a [Quaternion](#).

A confidence value is also available via the [getOrientationConfidence\(\)](#) function to indicate how sure the algorithm is of this value.

Returns:

Joint orientation

`float nite::SkeletonJoint::getOrientationConfidence() const [inline]`

Indicates how sure the [NiTE](#) skeleton algorithm is about the orientation data stored about this joint. The value is between 0 and 1, with increasing value indicating increasing confidence.

Returns:

Confidence value between 0 and 1

`const Point3f& nite::SkeletonJoint::getPosition() const [inline]`

Get the current position of the joint. This will be represented as a point in space. The coordinates used are "real world" coordinates. See the OpenNI2 documentation for more information on coordinate systems.

A confidence value is also available via the [getPositionConfidence\(\)](#) function to indicate how sure the algorithm is of this value.

Returns:

Position in space of the joint.

float nite::SkeletonJoint::getPositionConfidence() const [inline]

Indicates how sure the **NiTE** skeleton algorithm is about the position data stored about this joint. The value is between 0 and 1, with increasing value indicating increasing confidence.

Returns:

Confidence value between 0 and 1

JointType nite::SkeletonJoint::getType() const [inline]

Gets the type of the joint represented by this particular object.

Returns:

Type of the joint.

See also:

JointType enumeration for a list of all possible joint types.

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

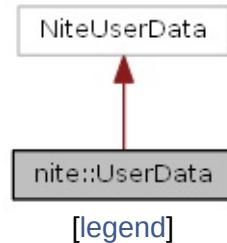
- **NiTE.h**



nite::UserData Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::UserData:



List of all members.

Public Member Functions

```
const BoundingBox & getBoundingBox () const
    const Point3f & getCenterOfMass () const
        UserId getId () const
    const PoseData & getPose (PoseType type) const
    const Skeleton & getSkeleton () const
        bool isLost () const
        bool isNew () const
        bool isVisible () const
```

Detailed Description

Provides the current information available about a specific user detected by the [UserTracker](#).

Note that some data is provided in depth ("projective") coordinates, and other data is provided in world ("real world") coordinates. See the OpenNI 2.0 documentation for information on the two coordinate systems used and how to convert between them.

See also:

[UserTracker](#) for more information on how this data is generated.

[Skeleton](#) for more information on a user's skeleton

[UserMap](#) for information on locating a specific user in the scene.

Member Function Documentation

`const BoundingBox& nite::UserData::getBoundingBox() const [i]`

Get a bounding box around the user. This provides a minimum volume cube that completely contains the user. Coordinates are "projective", so they are suitable for direct use with a depthmap.

Returns:

`BoundingBox` that surrounds the user in the `UserMap`.

`const Point3f& nite::UserData::getCenterOfMass() const [inline]`

Find the geometric center of mass of the user. This value is given in "real world" coordinates, so it must be converted before being superimposed over a raw depthmap. It is suitable for direct use with skeleton and hand point output.

Returns:

`Point3f` indicating position in space of the user.

`UserId nite::UserData::getId() const [inline]`

Get the ID of the user. This ID is persistent -- it will not change as long as the user is being continuously tracked.

Returns:

`UserId` of the user being tracked.

`const PoseData& nite::UserData::getPose(PoseType type) const`

Get all information about a specific pose for this user. This will only be available if pose detection is enabled on this user for the `PoseType`

specified.

Parameters:

[in] @ref PoseType to retrieve data for.

Returns:

[PoseData](#) for this pose type and this user.

See also:

[UserTracker](#) for more information on enabling Pose detected.

const [Skeleton&](#) nite::UserData::getSkeleton() const [inline]

Get the full skeleton of this user. This will only be available if [Skeleton](#) tracking has been enabled for this user, and the skeleton is calibrated and being actively tracked.

Returns:

[Skeleton](#) of the user.

See also:

[UserTracker](#) for information on enabling skeleton tracking.

bool nite::UserData::isLost() const [inline]

Check if the user is lost. This will happen once, in the first frame in which the user was declared lost. This user will not be provided in future frames, and his UserId may be assigned to a new user that appears in view.

Returns:

True if the user is no longer being tracked, False otherwise.

bool nite::UserData::isNew() const [inline]

Checks whether this user has been newly detected.

Returns:

True if this is the first frame this user appears in, False otherwise.

bool nite::UserData::isVisible () const [inline]

Check if the user is currently visible in the field of view.

Returns:

True if the user is visible and detected, false otherwise.

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

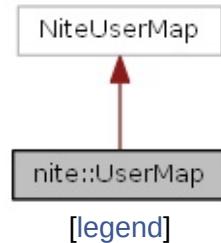
- [NiTE.h](#)



nite::UserMap Class Reference

```
#include <NiTE.h>
```

Inheritance diagram for nite::UserMap:



List of all members.

Public Member Functions

```
int getHeight () const  
const UserId * getPixels () const  
    int getStride () const  
    int getWidth () const
```

Friends

```
class UserTrackerFrameRef
```

Detailed Description

This class stores the output from the [nite::UserTracker](#) algorithm.

The basic purpose of the User Tracker algorithm is to analyze a depth frame and to locate all users in the scene, to indicate which pixels belong to which user, and which pixels belong to the background.

The output is in the form of a two dimensional array with the same dimensions as the depth frame that was used as the input to [UserTracker](#). Each element of the array corresponds to a single pixel from the depth map, and indicates the UserID of the user which occupies that pixel.

The UserId value 0 is used to indicate pixels that contain no users (ie background).

See also:

[UserTracker](#) for more information about this algorithm.

Member Function Documentation

`int nite::UserMap::getHeight() const [inline]`

Gets the height of the [UserMap](#) array. This value will match the Y resolution of the depth map that was used to create the [UserMap](#). This value can be used, along with the value returned by [getStride\(\)](#) to determine the size of the array in bytes.

Returns:

Height of the [UserMap](#), measured in pixels.

`const UserId* nite::UserMap::getPixels() const [inline]`

Returns a direct pointer to the array containing the [UserMap](#) data. This array has type UserID, and a size equal to $y \times \text{stride}$. If your application needs to calculate the size value directly, then see [UserMap::getHeight\(\)](#) and [UserMap::getStride\(\)](#).

Returns:

Pointer directly to the array data

`int nite::UserMap::getStride() const [inline]`

This function returns the stride of the array returned by `getPixels`. The stride of the array is defined as the width of a single row of that array, measured in bytes. This value can be multiplied by the value returned by [getHeight\(\)](#) to calculate the size of the array.

Returns:

Stride of the array, measured in bytes.

`int nite::UserMap::getWidth() const [inline]`

Gets the width of the [UserMap](#) array. This value will match the X resolution of the depth map that was used to create the [UserMap](#).

Returns:

Width of the [UserMap](#), measured in pixels.

Friends And Related Function Documentation

friend class UserTrackerFrameRef [friend]

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

- [NiTE.h](#)

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



nite::UserTracker

Class Reference

#include <[NiTE.h](#)>

List of all members.

Classes

class **NewFrameListener**

Public Member Functions

	UserTracker ()
	~UserTracker ()
void	addNewFrameListener (NewFrameListener *pListener)
Status	convertDepthCoordinatesToJoint (int x, int y, int z, float *pOutX, float *pOutY) const
Status	convertJointCoordinatesToDepth (float x, float y, float z, float *pOutX, float *pOutY) const
Status	create (openni::Device *pDevice=NULL)
void	destroy ()
float	getSkeletonSmoothingFactor () const
bool	isValid () const
Status	readFrame (UserTrackerFrameRef *pFrame)
void	removeNewFrameListener (NewFrameListener *pListener)
Status	setSkeletonSmoothingFactor (float factor)
Status	startPoseDetection (UserId user, PoseType type)
Status	startSkeletonTracking (UserId id)
void	stopPoseDetection (UserId user, PoseType type)
void	stopSkeletonTracking (UserId id)

Detailed Description

This is the main object of the User Tracker algorithm. It provides access to one half of the algorithms provided by [NiTE](#). Scene segmentation, skeleton, floor plane detection, and pose detection are all provided by this class.

The first purpose of the User Tracker algorithm is to find all of the active users in a specific scene. It individually tracks each human it finds, and provides the means to separate their outline from each other and from the background. Once the scene has been segmented, the User Tracker is also used to initiate [Skeleton](#) Tracking and Pose Detection algorithms.

Each user is provided an ID as they are detected. The user ID remains constant as long as the user remains in the frame. If a user leaves the field of view of the camera, or tracking of that user is otherwise lost, the user may have a different ID when he is detected again. There is currently no mechanism that provides persistent recognition of individuals when they are not being actively tracking. If this functionality is desired, it will need to be implemented at the application level.

A listener class is provided to allow event based interaction with this algorithm.

See also:

- [UserMap](#) for the output format of the User Tracker algorithm
 - [UserData](#) for additional data output by this format
 - [Skeleton](#) if you are also interested in tracking a user's skeleton
 - [NiTE](#) for a couple of static functions that must be run before User Tracker can be used
 - [HandTracker](#) for Gesture and Hand tracking algorithms.
-

Constructor & Destructor Documentation

`nite::UserTracker::UserTracker()` [inline]

Default constructor. Creates an empty [UserTracker](#) with a NULL handle. This object will not be useful until the [create\(\)](#) function is called.

See also:

[UserTracker::create\(\)](#) for a function to [create](#) and activate the algorithm.

[UserTracker::isValid\(\)](#) to determine whether [create\(\)](#) has already been called.

`nite::UserTracker::~UserTracker()` [inline]

Destructor. Automatically calls the provided [destroy\(\)](#) function.

Member Function Documentation

void nite::UserTracker::addNewFrameListener ([NewFrameListener](#)

Adds a NewFrameListner object to this [UserTracker](#) so that it will respond when a new frame is generated.

Parameters:

[in] **pListener** Pointer to a listener to add.

See also:

[UserTracker::NewFrameListener](#) for more information on using event based interaction with [UserTracker](#)

**Status nite::UserTracker::convertDepthCoordinatesToJoint (int
int
int
float *
float ***
)

In general, two coordinate systems are used in OpenNI 2.0. These conventions are also followed in [NiTE](#) 2.0.

[Skeleton](#) joint positions are provided in "Real World" coordinates, while the native coordinate system of depth maps is the "projective" system. In short, "Real World" coordinates locate objects using a Cartesian coordinate system with the origin at the sensor. "Projective" coordinates measure straight line distance from the sensor, and indicate x/y coordinates using pixels in the image (which is mathematically equivalent to specifying angles). See the OpenNI 2.0 documentation online for more information.

This function allows you to convert the native depth map coordinates

to the system used by the joints. This might be useful for performing certain types of measurements (eg distance between a joint and an object identified only in the depth map).

Note that no output is given for the Z coordinate. Z coordinates remain the same when performing the conversion. An input value is still required for Z, since this can affect the x/y output.

Parameters:

- [in] **x** The input X coordinate using the "projective" coordinate system.
- [in] **y** The input Y coordinate using the "projective" coordinate system.
- [in] **z** The input Z coordinate using the "projective" coordinate system.
- [out] **pOutX** Pointer to a location to store the output X coordinate in the "real world" system.
- [out] **pOutY** Pointer to a location to store the output Y coordinate in the "real world" system.

Returns:

Status indicating success or failure of this operation. This is needed because the ability to convert between coordinate systems requires a properly initialized Device from OpenNI 2.0.

```
Status nite::UserTracker::convertJointCoordinatesToDepth ( float
                                                               float
                                                               float
                                                               float *
                                                               float *
)

```

In general, two coordinate systems are used in OpenNI 2.0. These conventions are also followed in **NiTE** 2.0.

Skeleton joint positions are provided in "Real World" coordinates, while the native coordinate system of depth maps is the "projective" system. In short, "Real World" coordinates locate objects using a Cartesian coordinate system with the origin at the sensor. "Projective" coordinates measure straight line distance from the sensor (perpendicular to the sensor face), and indicate x/y coordinates using pixels in the image (which is mathematically equivalent to specifying angles). See the OpenNI 2.0 documentation online for more information.

Note that no output is given for the Z coordinate. Z coordinates remain the same when performing the conversion. An input value is still required for Z, since this can affect the x/y output.

This function allows you to convert the coordinates of a **SkeletonJoint** to the native coordinates of a depth map. This is useful if you need to find the joint position on the raw depth map.

Parameters:

- [in] **x** The input X coordinate using the "real world" coordinate system.
- [in] **y** The input Y coordinate using the "real world" coordinate system.
- [in] **z** The input Z coordinate using the "real world" coordinate system.
- [out] **pOutX** Pointer to a location to store the output X coordinate in the "projective" system.
- [out] **pOutY** Pointer to a location to store the output Y coordinate in the "projective" system.

Returns:

Status indicating success or failure of this operation. This is needed because the ability to convert between coordinate systems requires a properly initialized Device from OpenNI 2.0.

Status nite::UserTracker::create (openni::Device * pDevice = NULL)

Creates and initializes an empty User Tracker. This function should be the first one called when a new **UserTracker** object is constructed.

An OpenNI device with depth capabilities is required for this algorithm to work. See the OpenNI 2.0 documentation for more information about using an OpenNI 2.0 compliant hardware device and creating a Device object.

Parameters:

[in] **pDevice** A pointer to an initialized OpenNI 2.0 Device object that provides depth streams.

Returns:

A status code to indicate success/failure. Since this relies on an external hardware device, it is important for applications to check this value.

See also:

Status enumeration for a list of all possible status values generated by this call.

void nite::UserTracker::destroy() [inline]

Shuts down the user tracker and releases all resources used by it.

This is the opposite of **create()**. This function is called automatically by the destructor in the current implementation, but it is good practice to run it manually when the algorithm is no longer required. Running this function more than once is safe -- it simply exits if called on a non-valid **UserTracker**.

float nite::UserTracker::getSkeletonSmoothingFactor() const [in]

Queries the current skeleton smoothing factor.

Returns:

Current skeleton smoothing factor.

See also:

[setSkeletonSmoothingFactor](#) for more information on the smoothing factor, and the means to change it.

bool nite::UserTracker::isValid () const [inline]

Indicates whether the [UserTracker](#) is valid.

When a new [UserTracker](#) is first constructed, this function will indicate that it is invalid (ie return False). Once the [create\(\)](#) function has been successfully called, then this function will return True. If the [destroy\(\)](#) function is called, this function will again indicate invalid.

It is safe to run [create\(\)](#) and [destroy\(\)](#) without calling this function -- both of those functions already check this value and return without doing anything if no action is required.

Returns:

True if the [UserTracker](#) object is correctly initialized, False otherwise.

See also:

[create\(\)](#) function -- causes the [UserTracker](#) to become initialized.

[destroy\(\)](#) function -- causes the [UserTracker](#) to become uninitialized.

Status nite::UserTracker::readFrame (UserTrackerFrameRef * pFra

Gets the next snapshot of the algorithm. This causes all data to be generated for the next frame of the algorithm -- algorithm frames correspond to the input depth frames used to generate them.

Parameters:

pFrame [out] A pointer that will be set to point to the next frame of data.

Returns:

Status code indicating whether this operation was successful.

void nite::UserTracker::removeNewFrameListener ([NewFrameListener](#) pListener)

Removes a [NewFrameListener](#) object from this UserTracker's list of listeners. The listener will no longer respond when a new frame is generated.

Parameters:

[in] **pListener** Pointer to a listener to remove.

See also:

[UserTracker::NewFrameListener](#) for more information on using event based interaction with [UserTracker](#).

Status nite::UserTracker::setSkeletonSmoothingFactor (float factor)

Control the smoothing factor of the skeleton joints. Factor should be between 0 (no smoothing at all) and 1 (no movement at all).

Experimenting with this factor should allow you to fine tune the skeleton performance. Higher values will produce smoother operation of the skeleton, but may make the skeleton feel less responsive to the user.

Parameters:

[in] **factor** The smoothing factor.

Returns:

Status code indicating success or failure of this operation.

```
Status nite::UserTracker::startPoseDetection ( UserId user,
                                                PoseType type
                                              ) [inline]
```

This function commands the **UserTracker** to start detecting specific poses for a specific user.

Parameters:

[in] **user** The UserID of the user that you would like to detect a pose for.

[in] **type** The type of pose you would like to detect.

Returns:

Status code indicating success or failure of this operation.

See also:

PoseData For more information on pose detection and the output it generates.

PoseType enumeration for a list of the available poses that can be detected.

```
Status nite::UserTracker::startSkeletonTracking ( UserId id ) [inlin
```

Requests that the **Skeleton** algorithm starts tracking a specific user. Once started, the skeleton will provide information on the joint position and orientation for that user during each new frame of the **UserTracker**.

Note that the computational requirements of calculating a skeleton increase linearly with the number of users tracked. Tracking too many users may result in poor performance and high CPU utilization. If performance slows to the point where the skeleton is not calculated at the full frame rate of the depth data used to generate it, the algorithm tends to perform poorly.

Parameters:

[in] **UserID** The ID number of the user to calculate a skeleton for.

Returns:

Status code indicating success or failure of this operation.

See also:

[nite::Skeleton](#) for more information on the skeleton algorithm.

void nite::UserTracker::stopPoseDetection (**UserId **user**,
 PoseType **type**
) [inline]**

This function commands the pose detection algorithm to stop detecting a specific pose for a specific user. Since it is possible to detect multiple poses from multiple users, it is possible that detection of a different pose on the same user (or the same pose on a different user) may continue after this function is called.

Parameters:

[in] **user** The UserID of the user to stop detecting a specific pose for.
[in] **type** The **PoseType** of the pose to stop detecting.

void nite::UserTracker::stopSkeletonTracking (**UserId **id**) [inline]**

Stops skeleton tracking for a specific user. If multiple users are being tracked, this will only stop tracking for the user specified -- skeleton calculation will continue for remaining users.

Parameters:

[in] **UserID** of the person to stop tracking.

See also:

[**nite::Skeleton**](#) for more information on the skeleton algorithm.

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

- [**NiTE.h**](#)
-

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [**doxygen**](#) 1.7.5.1



NiTE 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Hierarchy](#)[Class Members](#)[nite](#) › [UserTrackerFrameRef](#) ›[Public Member Functions](#) | [Friends](#)

nite::UserTrackerFrameRef Class Reference

#include <[NiTE.h](#)>

List of all members.

Public Member Functions

	UserTrackerFrameRef ()
	UserTrackerFrameRef (const UserTrackerFrameRef &other)
	~UserTrackerFrameRef ()
openni::VideoFrameRef	getDepthFrame ()
const Plane &	getFloor () const
float	getFloorConfidence () const
int	getFrameIndex () const
uint64_t	getTimestamp () const
const UserData *	getUserById (UserId id) const
const UserMap &	getUserMap () const
const Array< UserData > &	getUsers () const
bool	isValid () const
UserTrackerFrameRef &	operator= (const UserTrackerFrameRef &other)
void	release ()

Friends

```
class User
```

```
class UserTracker
```

Detailed Description

This class stores a snapshot of a single frame of output from the [UserTracker](#) algorithm. This frame will correspond to a single input depth frame. It holds all info on users, as well as information on the floor plane.

See also:

[UserData](#) For detailed info on specific users

[Skeleton](#) For user skeleton data

[UserMap](#) For user positions in the frame

[UserTracker](#) For information on starting the user tracker algorithms to detect all of this data.

[HandTrackerFrameRef](#) for the equivalent information when hand tracking.

Constructor & Destructor Documentation

nite::UserTrackerFrameRef::UserTrackerFrameRef () [inline]

Default Constructor. Creates an empty **UserTrackerFrameRef**.

nite::UserTrackerFrameRef::~UserTrackerFrameRef () [inline]

Destructor. Frees the resources used by a **UserTrackerFrameRef**.

nite::UserTrackerFrameRef::UserTrackerFrameRef (const **UserTrac**

Copy Constructor. Creates a new **UserTrackerFrameRef** identical to an existing one.

Member Function Documentation

`openni::VideoFrameRef nite::UserTrackerFrameRef::getDepthFrame()`

Get the raw depth frame that originated this output. Each [UserTracker](#) frame is generated to correspond with a single depth frame. This function gives you access to the raw frame data.

See the OpenNI 2.0 documentation for information on manipulating VideoFrameRef data.

Returns:

[VideoFrameRef](#) with the raw depth that generated this frame.

`const Plane& nite::UserTrackerFrameRef::getFloor() const [inlin`

Obtains an estimate of the floor plane of the scene currently in view.

Returns:

[Plane](#) that indicates a best guess of where the floor of the scene is.

See also:

[getFloorConfidence\(\)](#) for a confidence value in this calculation.

`float nite::UserTrackerFrameRef::getFloorConfidence() const [in:`

[NiTE](#) generates a confidence value between 0 and 1 for the floor plane calculation. Lower values indicate less confidence.

Returns:

Value between 0 and 1 indicating confidence in the accuracy of the floor plane

See also:

[getFloor\(\)](#) to obtain the actual plane.

`int nite::UserTrackerFrameRef::getFrameIndex() const [inline]`

Gets the frame index of the raw depth frame that generated this [UserTracker](#) frame. Frame indexes are consequitive integer numbers assigned to frames by OpenNI. See the OpenNI documentation for more information.

Returns:

FrameIndex of the depth frame used to generate this User Tracker frame.

`uint64_t nite::UserTrackerFrameRef::getTimestamp() const [inlin`

Get the timestamp in which this frame was processed. See the OpenNI 2.0 documentation for more information on how OpenNI assigns time stamps to depth frames.

Returns:

Timestamp in microseconds

`const UserData* nite::UserTrackerFrameRef::getUserById(UserId`

Provides access to the [UserData](#) object for a specific user, indexed by [UserId](#).

Parameters:

`[in] @ref UserId` The ID of the user you would like data from.

Returns:

Pointer to the corresponding [UserData](#) object.

See also:

[UserData](#) for the information contained in this object.

[UserMap](#) if what you really want is to know where in the image the user is.

`const UserMap& nite::UserTrackerFrameRef::getUserMap() const`

Get the segmentation of the scene. The [UserTracker](#) algorithm locates each user in the field of view, and figures out which pixels correspond to which user, as well as which pixels correspond to the background (ie, not to any user).

Returns:

[UserMap](#) showing which pixels are occupied by what users.

`const Array<UserData>& nite::UserTrackerFrameRef::getUsers() const`

Get an [Array](#) of all the users available in this frame.

Returns:

[Array](#) of [UserData](#) with one entry for each user in this frame.

`bool nite::UserTrackerFrameRef::isValid() const [inline]`

Indicates whether this [UserTrackerFrameRef](#) points to valid frame data. When first constructed, the [UserTrackerFrameRef](#) will be invalid until assigned a frame handle.

Returns:

True if this object contains valid data, False otherwise.

`UserTrackerFrameRef& nite::UserTrackerFrameRef::operator=(const UserTrackerFrameRef &)`

Implements the assignment operator for the **UserTrackerFrameRef**.

Parameters:

[in] **other** A **UserTrackerFrameRef** to be assigned to this one.

void nite::UserTrackerFrameRef::release () [inline]

Properly releases all resources used by this object. This will be automatically called by the destructor, but it is good practice to call this as soon as the object is no longer required. Calling this function multiple times on the same object is safe.

Friends And Related Function Documentation

friend class User [friend]

friend class UserTracker [friend]

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

- [NiTE.h](#)

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



NiTE 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)[Class Hierarchy](#)[Class Members](#)

Class Index

[A](#) | [B](#) | [G](#) | [H](#) | [N](#) | [P](#) | [Q](#) | [S](#) | [U](#)[NiTE \(nite\)](#)[A](#)[H](#)[NiteBoundingBox](#)[Array \(nite\)](#)[HandData \(nite\)](#)[NitePlane](#)[HandTracker \(nite\)](#)[NitePoint3f](#)[B](#)[HandTrackerFrameRef \(nite\)](#)[NiteQuaternion](#)[BoundingBox
\(nite\)](#)[N](#)[P](#)[G](#)[UserTracker::NewFrameListener
\(nite\)](#)[Plane \(nite\)](#)[GestureData](#) [HandTracker::NewFrameListener
\(nite\)](#) [Point3f \(nite\)](#)[A](#) | [B](#) | [G](#) | [H](#) | [N](#) | [P](#) | [Q](#) | [S](#) | [U](#)



Class Hierarchy

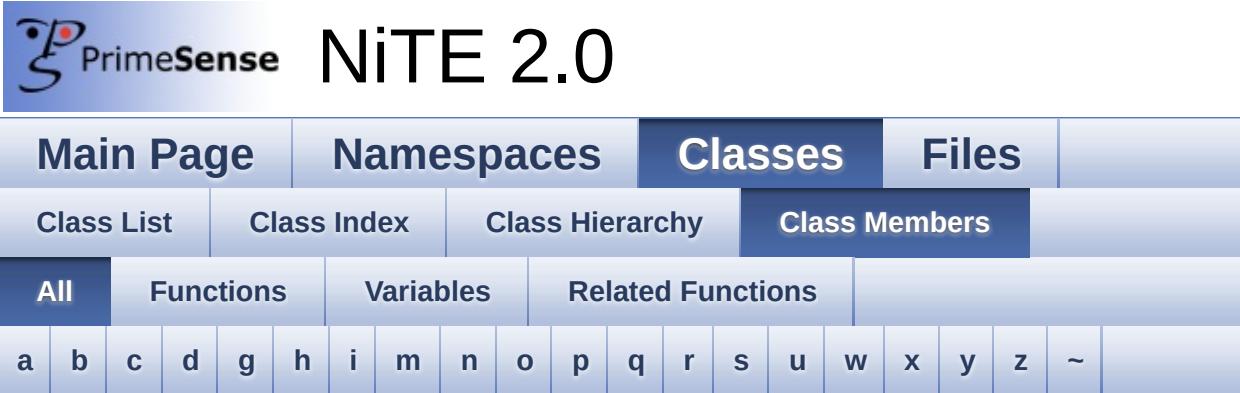
[Go to the graphical class hierarchy](#)

This inheritance list is sorted roughly, but not completely, alphabetically:

- [nite::Array< T >](#)
- [nite::HandTracker](#)
- [nite::HandTrackerFrameRef](#)
- [nite::HandTracker::NewFrameListener](#)
- [nite::UserTracker::NewFrameListener](#)
- [nite::NiTE](#)
- [NiteBoundingBox](#)
 - [nite::BoundingBox](#)
- [NiteGestureData](#)
 - [nite::GestureData](#)
- [NiteHandData](#)
 - [nite::HandData](#)
- [NitePlane](#)
 - [nite::Plane](#)
- [NitePoint3f](#)
 - [nite::Point3f](#)
- [NitePoseData](#)
 - [nite::PoseData](#)
- [NiteQuaternion](#)
 - [nite::Quaternion](#)
- [NiteSkeleton](#)
 - [nite::Skeleton](#)
- [NiteSkeletonJoint](#)
 - [nite::SkeletonJoint](#)

- **NiteUserData**
 - **nite::UserData**
 - **NiteUserMap**
 - **nite::UserMap**
 - **nite::UserTracker**
 - **nite::UserTrackerFrameRef**
-

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



The header features the PrimeSense logo (a stylized 'P' with a red eye) followed by the text "PrimeSense NiTE 2.0". Below this is a navigation menu with tabs: "Main Page", "Namespaces", "Classes" (which is highlighted in blue), and "Files". Under "Classes", there are sub-tabs: "Class List", "Class Index", "Class Hierarchy", and "Class Members". Further down are tabs for "All", "Functions", "Variables", and "Related Functions", with a scrollable list of letters (a, b, c, d, g, h, i, m, n, o, p, q, r, s, u, w, x, y, z, ~) below them.

Here is a list of all class members with links to the classes they belong to:

- a -

- addNewFrameListener() : [nite::UserTracker](#) , [nite::HandTracker](#)
- Array() : [nite::Array< T >](#)

- b -

- BoundingBox() : [nite::BoundingBox](#)

- c -

- convertDepthCoordinatesToHand() : [nite::HandTracker](#)
- convertDepthCoordinatesToJoint() : [nite::UserTracker](#)
- convertHandCoordinatesToDepth() : [nite::HandTracker](#)
- convertJointCoordinatesToDepth() : [nite::UserTracker](#)
- create() : [nite::UserTracker](#) , [nite::HandTracker](#)

- d -

- destroy() : [nite::UserTracker](#) , [nite::HandTracker](#)

- g -

- getBoundingBox() : [nite::UserData](#)
- getCenterOfMass() : [nite::UserData](#)
- getCurrentPosition() : [nite::GestureData](#)
- getDepthFrame() : [nite::UserTrackerFrameRef](#) ,

nite::HandTrackerFrameRef

- `getFloor() : nite::UserTrackerFrameRef`
- `getFloorConfidence() : nite::UserTrackerFrameRef`
- `getFrameIndex() : nite::UserTrackerFrameRef ,
nite::HandTrackerFrameRef`
- `getGestures() : nite::HandTrackerFrameRef`
- `getHands() : nite::HandTrackerFrameRef`
- `getHeight() : nite::UserMap`
- `getId() : nite::UserData , nite::HandData`
- `getJoint() : nite::Skeleton`
- `getOrientation() : nite::SkeletonJoint`
- `getOrientationConfidence() : nite::SkeletonJoint`
- `getPixels() : nite::UserMap`
- `getPose() : nite::UserData`
- `getPosition() : nite::SkeletonJoint , nite::HandData`
- `getPositionConfidence() : nite::SkeletonJoint`
- `getSize() : nite::Array< T >`
- `getSkeleton() : nite::UserData`
- `getSkeletonSmoothingFactor() : nite::UserTracker`
- `getSmoothingFactor() : nite::HandTracker`
- `getState() : nite::Skeleton`
- `getStride() : nite::UserMap`
- `getTimestamp() : nite::UserTrackerFrameRef ,
nite::HandTrackerFrameRef`
- `getType() : nite::SkeletonJoint , nite::PoseData ,
nite::GestureData`
- `getUserById() : nite::UserTrackerFrameRef`
- `getUserMap() : nite::UserTrackerFrameRef`
- `getUsers() : nite::UserTrackerFrameRef`
- `getVersion() : nite::NiTE`
- `getWidth() : nite::UserMap`

- h -

- `HandTracker : nite::HandTrackerFrameRef ,
nite::HandTracker::NewFrameListener , nite::HandTracker`
- `HandTrackerFrameRef() : nite::HandTrackerFrameRef`

- i -

- initialize() : **nite::NiTE**
- isComplete() : **nite::GestureData**
- isEmpty() : **nite::Array< T >**
- isEntered() : **nite::PoseData**
- isExited() : **nite::PoseData**
- isHeld() : **nite::PoseData**
- isInProgress() : **nite::GestureData**
- isLost() : **nite::UserData , nite::HandData**
- isNew() : **nite::HandData , nite::UserData**
- isTouchingFov() : **nite::HandData**
- isTracking() : **nite::HandData**
- isValid() : **nite::HandTrackerFrameRef , nite::UserTracker ,
nite::HandTracker , nite::UserTrackerFrameRef**
- isVisible() : **nite::UserData**

- m -

- max : **NiteBoundingBox**
- min : **NiteBoundingBox**

- n -

- NewFrameListener() : **nite::UserTracker::NewFrameListener ,
nite::HandTracker::NewFrameListener**
- normal : **NitePlane**

- o -

- onNewFrame() : **nite::UserTracker::NewFrameListener ,
nite::HandTracker::NewFrameListener**
- operator!=() : **nite::Point3f**
- operator==() : **nite::Point3f , nite::UserTrackerFrameRef ,
nite::HandTrackerFrameRef**
- operator==() : **nite::Point3f**
- operator[]() : **nite::Array< T >**

- p -

- Plane() : **nite::Plane**
- point : **NitePlane**
- Point3f() : **nite::Point3f**

- q -

- Quaternion() : **nite::Quaternion**

- r -

- readFrame() : **nite::UserTracker** , **nite::HandTracker**
- release() : **nite::HandTrackerFrameRef** ,
nite::UserTrackerFrameRef
- removeNewFrameListener() : **nite::HandTracker** ,
nite::UserTracker

- s -

- set() : **nite::Point3f**
- setData() : **nite::Array< T >**
- setSkeletonSmoothingFactor() : **nite::UserTracker**
- setSmoothingFactor() : **nite::HandTracker**
- shutdown() : **nite::NiTE**
- startGestureDetection() : **nite::HandTracker**
- startHandTracking() : **nite::HandTracker**
- startPoseDetection() : **nite::UserTracker**
- startSkeletonTracking() : **nite::UserTracker**
- stopGestureDetection() : **nite::HandTracker**
- stopHandTracking() : **nite::HandTracker**
- stopPoseDetection() : **nite::UserTracker**
- stopSkeletonTracking() : **nite::UserTracker**

- u -

- User : **nite::UserTrackerFrameRef**
- UserTracker : **nite::UserTrackerFrameRef** , **nite::UserTracker** ,
nite::UserTracker::NewFrameListener

- UserTrackerFrameRef() : **nite::UserTrackerFrameRef** ,
nite::UserMap , **nite::UserTrackerFrameRef**

- **W** -

- w : **NiteQuaternion**

- **X** -

- x : **NitePoint3f** , **NiteQuaternion**

- **y** -

- y : **NitePoint3f** , **NiteQuaternion**

- **Z** -

- z : **NitePoint3f** , **NiteQuaternion**

- ~ -

- ~HandTracker() : **nite::HandTracker**
- ~HandTrackerFrameRef() : **nite::HandTrackerFrameRef**
- ~UserTracker() : **nite::UserTracker**
- ~UserTrackerFrameRef() : **nite::UserTrackerFrameRef**

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Namespace List](#)[Namespace Members](#)

Namespace List

Here is a list of all namespaces with brief descriptions:

[nite](#)

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



NiTE 2.0

[Main Page](#)[Namespaces](#)[Classes](#)[Files](#)[Namespace List](#)[Namespace Members](#)[Classes](#) | [Typedefs](#) | [Enumerations](#)

nite Namespace Reference

Classes

class	Array
class	BoundingBox
class	GestureData
class	HandData
class	HandTracker
class	HandTrackerFrameRef
class	NiTE
class	Plane
class	Point3f
class	PoseData
class	Quaternion
class	Skeleton
class	SkeletonJoint
class	UserData
class	UserMap
class	UserTracker
class	UserTrackerFrameRef

TypeDefs

```
typedef short int HandId
```

```
typedef short int UserId
```

Enumerations

```
enum GestureType { GESTURE_WAVE, GESTURE_CLICK,
GESTURE_HAND_RAISE }

enum JointType {
    JOINT_HEAD, JOINT_NECK, JOINT_LEFT_SHOULDER,
JOINT_RIGHT_SHOULDER,
    JOINT_LEFT_ELBOW, JOINT_RIGHT_ELBOW,
JOINT_LEFT_HAND, JOINT_RIGHT_HAND,
    JOINT_TORSO, JOINT_LEFT_HIP, JOINT_RIGHT_HIP,
JOINT_LEFT_KNEE,
    JOINT_RIGHT_KNEE, JOINT_LEFT_FOOT,
JOINT_RIGHT_FOOT
}

enum PoseType { POSE_PSI, POSE_CROSSED_HANDS }

enum SkeletonState {
    SKELETON_NONE, SKELETON_CALIBRATING,
SKELETON_TRACKED,
    SKELETON_CALIBRATION_ERROR_NOT_IN_POSE,
    SKELETON_CALIBRATION_ERROR_HANDS,
SKELETON_CALIBRATION_ERROR_HEAD,
    SKELETON_CALIBRATION_ERROR_LEGS,
SKELETON_CALIBRATION_ERROR_TORSO
}

enum Status { STATUS_OK, STATUS_ERROR,
STATUS_BAD_USER_ID, STATUS_OUT_OF_FLOW }
```

TypeDef Documentation

typedef short int nite::HandId

typedef short int nite::UserId

UserId is a persistent ID for a specific user detected by the [UserTracker](#) algorithm. While the user is known, it will have the same ID.

Enumeration Type Documentation

enum nite::GestureType

Available gestures types

Enumerator:

GESTURE_WAVE
GESTURE_CLICK
GESTURE_HAND_RAISE

enum nite::JointType

Available joints in skeleton

Enumerator:

JOINT_HEAD
JOINT_NECK
JOINT_LEFT_SHOULDER
JOINT_RIGHT_SHOULDER
JOINT_LEFT_ELBOW
JOINT_RIGHT_ELBOW
JOINT_LEFT_HAND
JOINT_RIGHT_HAND
JOINT_TORSO
JOINT_LEFT_HIP
JOINT_RIGHT_HIP
JOINT_LEFT_KNEE
JOINT_RIGHT_KNEE
JOINT_LEFT_FOOT
JOINT_RIGHT_FOOT

enum nite::PoseType

Enumerator:

POSE_PSI

POSE_CROSSED_HANDS

enum nite::SkeletonState

Possible states of the skeleton

Enumerator:

SKELETON_NONE

No
skeleton -
skeleton
was not
requested

SKELETON_CALIBRATING

Skeleton
requested
but still
unavailabl

SKELETON_TRACKED

Skeleton
available

SKELETON_CALIBRATION_ERROR_NOT_IN_POSE

Possible
reasons a:
to why
skeleton is
unavailabl

SKELETON_CALIBRATION_ERROR_HANDS

```
SKELETON_CALIBRATION_ERROR_HEAD  
SKELETON_CALIBRATION_ERROR_LEGS  
SKELETON_CALIBRATION_ERROR_TORSO
```

enum nite::Status

Possible failure values

Enumerator:

```
STATUS_OK  
STATUS_ERROR  
STATUS_BAD_USER_ID  
STATUS_OUT_OF_FLOW
```



NiTE 2.0

Main Page	Namespaces	Classes	Files	
Namespace List	Namespace Members			
All	Typedefs	Enumerations	Enumerator	
g	h	j	p	s

Here is a list of all namespace members with links to the namespace documentation for each member:

- g -

- GESTURE_CLICK : [nite](#)
- GESTURE_HAND_RAISE : [nite](#)
- GESTURE_WAVE : [nite](#)
- GestureType : [nite](#)

- h -

- HandId : [nite](#)

- j -

- JOINT_HEAD : [nite](#)
- JOINT_LEFT_ELBOW : [nite](#)
- JOINT_LEFT_FOOT : [nite](#)
- JOINT_LEFT_HAND : [nite](#)
- JOINT_LEFT_HIP : [nite](#)
- JOINT_LEFT_KNEE : [nite](#)
- JOINT_LEFT_SHOULDER : [nite](#)
- JOINT_NECK : [nite](#)
- JOINT_RIGHT_ELBOW : [nite](#)
- JOINT_RIGHT_FOOT : [nite](#)
- JOINT_RIGHT_HAND : [nite](#)
- JOINT_RIGHT_HIP : [nite](#)
- JOINT_RIGHT_KNEE : [nite](#)

- JOINT_RIGHT_SHOULDER : **nite**
- JOINT_TORSO : **nite**
- JointType : **nite**

- p -

- POSE_CROSSED_HANDS : **nite**
- POSE_PSI : **nite**
- PoseType : **nite**

- s -

- SKELETON_CALIBRATING : **nite**
- SKELETON_CALIBRATION_ERROR_HANDS : **nite**
- SKELETON_CALIBRATION_ERROR_HEAD : **nite**
- SKELETON_CALIBRATION_ERROR_LEGS : **nite**
- SKELETON_CALIBRATION_ERROR_NOT_IN_POSE : **nite**
- SKELETON_CALIBRATION_ERROR_TORSO : **nite**
- SKELETON_NONE : **nite**
- SKELETON_TRACKED : **nite**
- SkeletonState : **nite**
- Status : **nite**
- STATUS_BAD_USER_ID : **nite**
- STATUS_ERROR : **nite**
- STATUS_OK : **nite**
- STATUS_OUT_OF_FLOW : **nite**

- u -

- UserId : **nite**



File List

Here is a list of all files with brief descriptions:

NiTE.h [code]	
---------------	--

NiteCTypes.h [code]	
---------------------	--

NiteEnums.h [code]	
--------------------	--



NiTE.h File Reference

```
#include "NiteCAPI.h" #include <OpenNI.h>
#include "NiteEnums.h"
```

[Go to the source code of this file.](#)

Classes

```
class nite::Array< T >
class nite::BoundingBox
class nite::GestureData
class nite::HandData
class nite::HandTracker
class nite::HandTrackerFrameRef
class nite::HandTracker::NewFrameListener
class nite::UserTracker::NewFrameListener
class nite::NiTE
class nite::Plane
class nite::Point3f
class nite::PoseData
class nite::Quaternion
class nite::Skeleton
class nite::SkeletonJoint
class nite::UserData
class nite::UserMap
class nite::UserTracker
class nite::UserTrackerFrameRef
```

Namespaces

```
namespace nite
```

TypeDefs

```
typedef short int nite::Handle
```

```
typedef short int nite::UserId
```

Enumerations

```
enum nite::GestureType { nite::GESTURE_WAVE,
    nite::GESTURE_CLICK, nite::GESTURE_HAND_RAISE }

enum nite::JointType {
    nite::JOINT_HEAD, nite::JOINT_NECK,
    nite::JOINT_LEFT_SHOULDER,
    nite::JOINT_RIGHT_SHOULDER,
    nite::JOINT_LEFT_ELBOW, nite::JOINT_RIGHT_ELBOW,
    nite::JOINT_LEFT_HAND, nite::JOINT_RIGHT_HAND,
    nite::JOINT_TORSO, nite::JOINT_LEFT_HIP,
    nite::JOINT_RIGHT_HIP, nite::JOINT_LEFT_KNEE,
    nite::JOINT_RIGHT_KNEE, nite::JOINT_LEFT_FOOT,
    nite::JOINT_RIGHT_FOOT
}

enum nite::PoseType { nite::POSE_PSI,
    nite::POSE_CROSSED_HANDS }

enum nite::SkeletonState {
    nite::SKELETON_NONE,
    nite::SKELETON_CALIBRATING,
    nite::SKELETON_TRACKED,
    nite::SKELETON_CALIBRATION_ERROR_NOT_IN_POSE,
    nite::SKELETON_CALIBRATION_ERROR_HANDS,
    nite::SKELETON_CALIBRATION_ERROR_HEAD,
    nite::SKELETON_CALIBRATION_ERROR_LEGS,
    nite::SKELETON_CALIBRATION_ERROR_TORSO
}

enum nite::Status { nite::STATUS_OK, nite::STATUS_ERROR,
    nite::STATUS_BAD_USER_ID,
    nite::STATUS_OUT_OF_FLOW }
```



NiteCTypes.h File Reference

```
#include <OniCTypes.h> #include "NiteCEnums.h"
```

[Go to the source code of this file.](#)

Classes

```
struct NiteBoundingBox  
struct NitePlane  
struct NitePoint3f  
struct NiteQuaternion
```

Defines

```
#define NITE_JOINT_COUNT 15  
#define NITE_POSE_COUNT 2
```

TypeDefs

```
typedef short int NiteHandId  
typedef struct NiteHandTracker * NiteHandTrackerHandle  
typedef short int NiteUserId  
typedef struct NiteUserTracker * NiteUserTrackerHandle
```

Define Documentation

```
#define NITE_JOINT_COUNT 15
```

```
#define NITE_POSE_COUNT 2
```

Typedef Documentation

typedef short int NiteHandId

typedef struct NiteHandTracker* NiteHandTrackerHandle

typedef short int NiteUserId

typedef struct NiteUserTracker* NiteUserTrackerHandle

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



NiteEnums.h File Reference

[Go to the source code of this file.](#)

Enumerations

```
enum GestureType { GESTURE_WAVE, GESTURE_CLICK,
GESTURE_HAND_RAISE }

enum JointType {
    JOINT_HEAD, JOINT_NECK, JOINT_LEFT_SHOULDER,
JOINT_RIGHT_SHOULDER,
    JOINT_LEFT_ELBOW, JOINT_RIGHT_ELBOW,
JOINT_LEFT_HAND, JOINT_RIGHT_HAND,
    JOINT_TORSO, JOINT_LEFT_HIP, JOINT_RIGHT_HIP,
JOINT_LEFT_KNEE,
    JOINT_RIGHT_KNEE, JOINT_LEFT_FOOT,
JOINT_RIGHT_FOOT
}

enum PoseType { POSE_PSI, POSE_CROSSED_HANDS }

enum SkeletonState {
    SKELETON_NONE, SKELETON_CALIBRATING,
SKELETON_TRACKED,
    SKELETON_CALIBRATION_ERROR_NOT_IN_POSE,
    SKELETON_CALIBRATION_ERROR_HANDS,
SKELETON_CALIBRATION_ERROR_HEAD,
    SKELETON_CALIBRATION_ERROR_LEGS,
SKELETON_CALIBRATION_ERROR_TORSO
}

enum Status { STATUS_OK, STATUS_ERROR,
STATUS_BAD_USER_ID, STATUS_OUT_OF_FLOW }
```

Enumeration Type Documentation

enum GestureType

Available gestures types

Enumerator:

GESTURE_WAVE
GESTURE_CLICK
GESTURE_HAND_RAISE

enum JointType

Available joints in skeleton

Enumerator:

JOINT_HEAD
JOINT_NECK
JOINT_LEFT_SHOULDER
JOINT_RIGHT_SHOULDER
JOINT_LEFT_ELBOW
JOINT_RIGHT_ELBOW
JOINT_LEFT_HAND
JOINT_RIGHT_HAND
JOINT_TORSO
JOINT_LEFT_HIP
JOINT_RIGHT_HIP
JOINT_LEFT_KNEE
JOINT_RIGHT_KNEE
JOINT_LEFT_FOOT
JOINT_RIGHT_FOOT

enum PoseType

Enumerator:

POSE_PSI

POSE_CROSSED_HANDS

enum SkeletonState

Possible states of the skeleton

Enumerator:

SKELETON_NONE

No
skeleton -
skeleton
was not
requested

SKELETON_CALIBRATING

Skeleton
requested
but still
unavailabl

SKELETON_TRACKED

Skeleton
available

SKELETON_CALIBRATION_ERROR_NOT_IN_POSE

Possible
reasons as
to why
skeleton is
unavailabl

SKELETON_CALIBRATION_ERROR_HANDS

```
SKELETON_CALIBRATION_ERROR_HEAD  
SKELETON_CALIBRATION_ERROR_LEGS  
SKELETON_CALIBRATION_ERROR_TORSO
```

enum Status

Possible failure values

Enumerator:

```
STATUS_OK  
STATUS_ERROR  
STATUS_BAD_USER_ID  
STATUS_OUT_OF_FLOW
```



Navigation menu:

- Main Page
- Namespaces
- Classes
- Files

File Members sub-menu:

- All
- Typedefs
- Enumerations
- Enumerator
- Defines

File members index:

- g
- j
- n
- p
- s

Here is a list of all file members with links to the files they belong to:

- g -

- GESTURE_CLICK : [NiteEnums.h](#)
- GESTURE_HAND_RAISE : [NiteEnums.h](#)
- GESTURE_WAVE : [NiteEnums.h](#)
- GestureType : [NiteEnums.h](#)

- j -

- JOINT_HEAD : [NiteEnums.h](#)
- JOINT_LEFT_ELBOW : [NiteEnums.h](#)
- JOINT_LEFT_FOOT : [NiteEnums.h](#)
- JOINT_LEFT_HAND : [NiteEnums.h](#)
- JOINT_LEFT_HIP : [NiteEnums.h](#)
- JOINT_LEFT_KNEE : [NiteEnums.h](#)
- JOINT_LEFT_SHOULDER : [NiteEnums.h](#)
- JOINT_NECK : [NiteEnums.h](#)
- JOINT_RIGHT_ELBOW : [NiteEnums.h](#)
- JOINT_RIGHT_FOOT : [NiteEnums.h](#)
- JOINT_RIGHT_HAND : [NiteEnums.h](#)
- JOINT_RIGHT_HIP : [NiteEnums.h](#)
- JOINT_RIGHT_KNEE : [NiteEnums.h](#)
- JOINT_RIGHT_SHOULDER : [NiteEnums.h](#)
- JOINT_TORSO : [NiteEnums.h](#)
- JointType : [NiteEnums.h](#)

- n -

- NITE_JOINT_COUNT : [NiteCTypes.h](#)
- NITE_POSE_COUNT : [NiteCTypes.h](#)
- NiteHandId : [NiteCTypes.h](#)
- NiteHandTrackerHandle : [NiteCTypes.h](#)
- NiteUserId : [NiteCTypes.h](#)
- NiteUserTrackerHandle : [NiteCTypes.h](#)

- p -

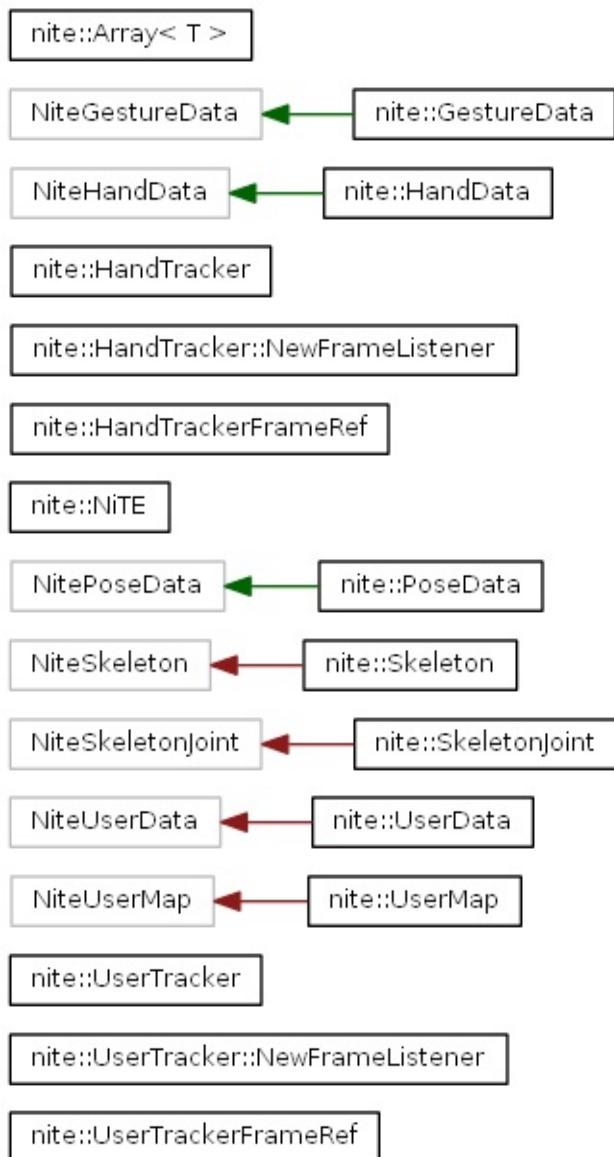
- POSE_CROSSED_HANDS : [NiteEnums.h](#)
- POSE_PSI : [NiteEnums.h](#)
- PoseType : [NiteEnums.h](#)

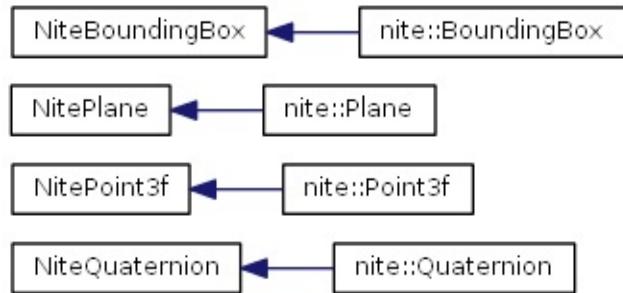
- s -

- SKELETON_CALIBRATING : [NiteEnums.h](#)
- SKELETON_CALIBRATION_ERROR_HANDS : [NiteEnums.h](#)
- SKELETON_CALIBRATION_ERROR_HEAD : [NiteEnums.h](#)
- SKELETON_CALIBRATION_ERROR_LEGS : [NiteEnums.h](#)
- SKELETON_CALIBRATION_ERROR_NOT_IN_POSE : [NiteEnums.h](#)
- SKELETON_CALIBRATION_ERROR_TORSO : [NiteEnums.h](#)
- SKELETON_NONE : [NiteEnums.h](#)
- SKELETON_TRACKED : [NiteEnums.h](#)
- SkeletonState : [NiteEnums.h](#)
- Status : [NiteEnums.h](#)
- STATUS_BAD_USER_ID : [NiteEnums.h](#)
- STATUS_ERROR : [NiteEnums.h](#)
- STATUS_OK : [NiteEnums.h](#)
- STATUS_OUT_OF_FLOW : [NiteEnums.h](#)

Class Hierarchy

Go to the textual class hierarchy





Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



NiTE.h

Go to the documentation of this file.

```
00001 /*****
* ****
00002 *
*
00003 *      PrimeSense NiTE 2.0
*
00004 *      Copyright (C) 2012 PrimeSense Ltd.
*
00005 *
*
00006 *****/
00007
00008 #ifndef _NITE_H_
00009 #define _NITE_H_
00010
00011 #include "NiteCAPI.h"
00012 #include <OpenNI.h>
00013
00014 // Summary of use cases, modules, facades
00015
00016 namespace nite {
00017 #include "NiteEnums.h"
00018
00019 // General
00020 _NITE_DECLARE_VERSION(Version);
00021
```

```
00027 class Point3f : public NitePoint3f
00028 {
00029 public:
00030     Point3f()
00031     {
00032         x = y = z = 0.0f;
00033     }
00034
00035     Point3f(float x, float y, float z)
00036     {
00037         this->set(x, y, z);
00038     }
00039
00040     Point3f(const Point3f& other)
00041     {
00042         *this = other;
00043     }
00044
00045     void set(float x, float y, float z)
00046     {
00047         this->x = x;
00048         this->y = y;
00049         this->z = z;
00050     }
00051
00052     Point3f& operator=(const Point3f& other)
00053     {
00054         set(other.x, other.y, other.z);
00055
00056         return *this;
00057     }
00058
00059     bool operator==(const Point3f& other) co
00060 nst
00061     {
00062         return x == other.x && y == other.y
00063 && z == other.z;
```

```
00095     }
00096
00103     bool operator!=(const Point3f& other) co
nst
00104     {
00105         return !operator==(other);
00106     }
00107 };
00108
00113 class Plane : public NitePlane
00114 {
00115     public:
00121     Plane()
00122     {
00123         this->point = Point3f();
00124         this->normal = Point3f();
00125     }
00126
00133     Plane(const Point3f& point, const Point3f
& normal)
00134     {
00135         this->point = point;
00136         this->normal = normal;
00137     }
00138 };
00139
00144 class Quaternion : public NiteQuaternion
00145 {
00146     public:
00151     Quaternion()
00152     {
00153         x = y = z = w = 0;
00154     }
00155
00164     Quaternion(float w, float x, float y, fl
oat z)
00165     {
```

```
00166         this->x = x;
00167         this->y = y;
00168         this->z = z;
00169         this->w = w;
00170     }
00171 };
00172
00177 class BoundingBox : public NiteBoundingBox
00178 {
00179 public:
00183     BoundingBox()
00184     {}
00185
00193     BoundingBox(const Point3f& min, const Po
int3f& max)
00194     {
00195         this->min = min;
00196         this->max = max;
00197     }
00198 };
00199
00204 template <class T>
00205 class Array
00206 {
00207 public:
00211     Array() : m_size(0), m_data(NULL) {}
00219     void setData(int size, T* data) {m_data
= data; m_size = size;}
00223     const T& operator[](int index) const {re
turn m_data[index];}
00228     int getSize() const {return m_size;}
00233     bool isEmpty() const {return m_size == 0
;}
00234 private:
00235     Array(const Array&);
00236     Array& operator=(const Array&);
00237
```

```
00238     int m_size;
00239     T* m_data;
00240 };
00241
00242 // UserTracker
00247 typedef short int UserId;
00248
00259 class PoseData : protected NitePoseData
00260 {
00261 public:
00267     PoseType getType() const {return (PoseTy
pe)type;}
00268
00274     bool isHeld() const {return (state & NIT
E_POSE_STATE_IN_POSE) != 0;}
00275
00281     bool isEntered() const {return (state &
NITE_POSE_STATE_ENTER) != 0;}
00282
00288     bool isExited() const {return (state & N
ITE_POSE_STATE_EXIT) != 0;}
00289 };
00290
00307 class UserMap : private NiteUserMap
00308 {
00309 public:
00318     const UserId* getPixels() const {return
pixels;}
00319
00326     int getWidth() const {return width;}
00327
00335     int getHeight() const {return height;}
00336
00344     int getStride() const {return stride;}
00345
00346     friend class UserTrackerFrameRef;
00347 };
```

```
00348
00358 class SkeletonJoint : private NiteSkeletonJo
int
00359 {
00360 public:
00368     JointType getType() const {return (Joint
Type)jointType;}
00369
00380     const Point3f& getPosition() const {retu
rn (Point3f&)position;}
00381
00389     float getPositionConfidence() const {ret
urn positionConfidence;}
00390
00400     const Quaternion& getOrientation() const
{return (Quaternion&)orientation;}
00401
00409     float getOrientationConfidence() const {
return orientationConfidence;}
00410 };
00435 class Skeleton : private NiteSkeleton
00436 {
00437 public:
00447     const SkeletonJoint& getJoint(JointType
type) const {return (SkeletonJoint&)joints[type];}
00457     SkeletonState getState() const {return (
SkeletonState)state;}
00458 };
00459
00460
00472 class UserData : private NiteUserData
00473 {
00474 public:
00481     UserId getId() const {return id;}
00482
00490     const BoundingBox& getBoundingBox() cons
t {return (const BoundingBox&)boundingBox;}
```

```
00491
00499     const Point3f& getCenterOfMass() const {
00500         return (const Point3f&)centerOfMass;
00500
00506     bool isNew() const {return (state & NITE
00507         _USER_STATE_NEW) != 0;}
00507
00513     bool isVisible() const {return (state &
00514         NITE_USER_STATE_VISIBLE) != 0;}
00514
00522     bool isLost() const {return (state & NIT
00523         E_USER_STATE_LOST) != 0;}
00523
00531     const Skeleton& getSkeleton() const {ret
00532         urn (const Skeleton&)skeleton;}
00532
00541     const PoseData& getPose(PoseType type) c
00542         onst {return (const PoseData&)poses[type];}
00542 };
00543
00554 class UserTrackerFrameRef
00555 {
00556     public:
00560     UserTrackerFrameRef() : m_pFrame(NULL),
00561         m_userTrackerHandle(NULL)
00561     {}
00562
00566     ~UserTrackerFrameRef()
00567     {
00568         release();
00569     }
00570
00575     UserTrackerFrameRef(const UserTrackerFra
00576         meRef& other) : m_pFrame(NULL)
00576     {
00577         *this = other;
00578     }
```

```
00579
00580     UserTrackerFrameRef& operator=(const Use
rTrackerFrameRef& other)
00581     {
00582         setReference(other.m_userTrackerHandle,
00583                     other.m_pFrame);
00584         niteUserTrackerFrameAddRef(m_userTra
ckerHandle, m_pFrame);
00585
00586         return *this;
00587     }
00588
00589     bool isValid() const
00590     {
00591         return m_pFrame != NULL;
00592     }
00593
00594     void release()
00595     {
00596         if (m_pFrame != NULL)
00597         {
00598             niteUserTrackerFrameRelease(m_us
erTrackerHandle, m_pFrame);
00599         }
00600         m_pFrame = NULL;
00601         m_userTrackerHandle = NULL;
00602     }
00603
00604     const UserData* getUserById(UserId id) c
onst
00605     {
00606         for (int i = 0; i < m_users.getSize(
); ++i)
00607         {
00608             if (m_users[i].getId() == id)
00609             {
00610                 return &m_users[i];
00611             }
00612         }
00613     }
00614
00615     void addUserData(const UserData& user)
00616     {
00617         m_users.push_back(user);
00618     }
00619
00620     void removeUserData(UserId id)
00621     {
00622         for (int i = 0; i < m_users.getSize();
++i)
00623         {
00624             if (m_users[i].getId() == id)
00625             {
00626                 m_users.erase(m_users.begin() + i);
00627                 break;
00628             }
00629         }
00630     }
00631
00632     void clear()
00633     {
00634         m_users.clear();
00635     }
00636 }
```

```
00636             }
00637         }
00638     return NULL;
00639 }
00640
00641     const Array<UserData>& getUsers() const {
00642 return m_users;}
00643
00644     float getFloorConfidence() const {return
00645     m_pFrame->floorConfidence;}
00646
00647     const Plane& getFloor() const {return (c
00648 onst Plane&)m_pFrame->floor;}
00649
00650     openni::VideoFrameRef getDepthFrame() {r
00651 eturn m_depthFrame;}
00652
00653     const UserMap& getUserMap() const {return
00654     static_cast<const UserMap&>(m_pFrame->userMap);}
00655
00656     uint64_t getTimestamp() const {return m_
00657 pFrame->timestamp;}
00658
00659     int getFrameIndex() const {return m_pFra
00660 me->frameIndex;}
00661 private:
00662     friend class User;
00663     friend class UserTracker;
00664
00665     Array<UserData> m_users;
00666
00667     void setReference(NiteUserTrackerHandle
00668 userTrackerHandle, NiteUserTrackerFrame* pFrame)
00669     {
00670         release();
00671         m_userTrackerHandle = userTrackerHan
00672 dle;
```

```
00709         m_pFrame = pFrame;
00710         m_depthFrame._ setFrame(pFrame->pDepth
hFrame);
00711         m_users.setData(m_pFrame->userCount,
(UserData*)m_pFrame->pUser);
00712     }
00713
00714
00715     NiteUserTrackerFrame* m_pFrame;
00716     NiteUserTrackerHandle m_userTrackerHandle;
00717     openni::VideoFrameRef m_depthFrame;
00718 };
00719
00745 class UserTracker
00746 {
00747 public:
00762     class NewFrameListener
00763     {
00764     public:
00769         NewFrameListener() : m_pUserTracker(
NULL)
00770         {
00771             m_userTrackerCallbacks.readyForN
extFrame = newFrameCallback;
00772         }
00773
00780         virtual void onNewFrame(UserTracker&
) = 0;
00781
00782     private:
00783         NiteUserTrackerCallbacks m_userTrack
erCallbacks;
00784
00785         NiteUserTrackerCallbacks& getCallbac
ks() {return m_userTrackerCallbacks;}
00786
```

```
00787         static void ONI_CALLBACK_TYPE newFra
meCallback(void* pCookie)
00788     {
00789         NewFrameListener* pListener = (N
ewFrameListener*)pCookie;
00790         pListener->onNewFrame(*pListener
->m_pUserTracker);
00791     }
00792
00793
00794     friend class UserTracker;
00795     void setUserTracker(UserTracker* pUs
erTracker)
00796     {
00797         m_pUserTracker = pUserTracker;
00798     }
00799
00800     UserTracker* m_pUserTracker;
00801 };
00802
00810     UserTracker() : m_userTrackerHandle(NULL
)
00811 {}
00812
00816     ~UserTracker()
00817     {
00818         destroy();
00819     }
00820
00835     Status create(openni::Device* pDevice =
NULL)
00836     {
00837         if (isValid())
00838         {
00839             // tracker already active
00840             return STATUS_OUT_OF_FLOW;
00841         }

```

```
00842
00843         if (pDevice == NULL)
00844     {
00845             return (Status)niteInitializeUser
rTracker(&m_userTrackerHandle);
00846     }
00847     return (Status)niteInitializeUserTra
ckerByDevice(pDevice, &m_userTrackerHandle);
00848 }
00849
00850     void destroy()
00851 {
00852     if (isValid())
00853     {
00854         niteShutdownUserTracker(m_userTr
ackerHandle);
00855         m_userTrackerHandle = NULL;
00856     }
00857 }
00858
00859     Status readFrame(UserTrackerFrameRef* pF
rame)
00860 {
00861     NiteUserTrackerFrame *pNiteFrame = N
ULL;
00862     Status rc = (Status)niteReadUserTrac
kerFrame(m_userTrackerHandle, &pNiteFrame);
00863     pFrame->setReference(m_userTrackerHa
ndle, pNiteFrame);
00864
00865     return rc;
00866 }
00867
00868     bool isValid() const
00869 {
00870     return m_userTrackerHandle != NULL;
00871 }
```

```
00902
00912     Status setSkeletonSmoothingFactor(float
factor)
00913     {
00914         return (Status)niteSetSkeletonSmooth
ing(m_userTrackerHandle, factor);
00915     }
00916
00924     float getSkeletonSmoothingFactor() const
00925     {
00926         float factor;
00927         Status rc = (Status)niteGetSkeletons
moothing(m_userTrackerHandle, &factor);
00928         if (rc != STATUS_OK)
00929         {
00930             factor = 0;
00931         }
00932         return factor;
00933     }
00934
00950     Status startSkeletonTracking(UserId id)
00951     {
00952         return (Status)niteStartSkeletonTrac
king(m_userTrackerHandle, id);
00953     }
00954
00963     void stopSkeletonTracking(UserId id)
00964     {
00965         niteStopSkeletonTracking(m_userTrack
erHandle, id);
00966     }
00967
00978     Status startPoseDetection(UserId user, P
oseType type)
00979     {
00980         return (Status)niteStartPoseDetectio
n(m_userTrackerHandle, (NiteUserId)user, (NitePose
```

```
0091     Type)type);
0092 }
0093
0094     void stopPoseDetection(UserId user, Pose
0095     Type type)
0096     {
0097         niteStopPoseDetection(m_userTrackerH
0098         andle, (NiteUserId)user, (NitePoseType)type);
0099     }
0100
0101     void addNewFrameListener(NewFrameListener
0102     * pListener)
0103     {
0104         niteRegisterUserTrackerCallbacks(m_u
0105         serTrackerHandle, &pListener->getCallbacks(), pLis
0106         tener);
0107         pListener->setUserTracker(this);
0108     }
0109
0110
0111     void removeNewFrameListener(NewFrameList
0112     ener* pListener)
0113     {
0114         niteUnregisterUserTrackerCallbacks(m
0115         _userTrackerHandle, &pListener->getCallbacks());
0116         pListener->setUserTracker(NULL);
0117     }
0118
0119
0120     Status convertJointCoordinatesToDepth(fl
0121     oat x, float y, float z, float* pOutX, float* pOut
0122     Y) const
0123     {
0124         return (Status)niteConvertJointCoord
0125         inatesToDepth(m_userTrackerHandle, x, y, z, pOutX,
0126         pOutY);
0127     }
0128
0129
0130     Status convertDepthCoordinatesToJoint(int
```

```
x, int y, int z, float* pOutX, float* pOutY) const

01075     {
01076         return (Status)niteConvertDepthCoord
01077             inatesToJoint(m_userTrackerHandle, x, y, z, pOutX,
01078                         pOutY);
01079     }
01080
01081 private:
01082     NiteUserTrackerHandle m_userTrackerHandle;
01083
01084 // HandTracker
01085 typedef short int HandId;
01086
01087 class GestureData : protected NiteGestureDat
01088 a
01089 {
01090 public:
01091     GestureType getType() const {return (Ges
01092 tureType)type;}
01093
01094     const Point3f& getCurrentPosition() cons
01095 t {return (Point3f&)currentPosition;}
01096
01097     bool isComplete() const {return (state &
01098         NITE_GESTURE_STATE_COMPLETED) != 0;}
01099
01100     bool isInProgress() const {return (state
01101         & NITE_GESTURE_STATE_IN_PROGRESS) != 0;}
01102
01103 };
01104
01105 class HandData : protected NiteHandData
01106 {
01107 public:
```

```
01151     HandId getId() const {return id;}
01152
01158     const Point3f& getPosition() const {retu
rn (Point3f&)position;}
01159
01165     bool isNew() const {return (state & NITE
_HAND_STATE_NEW) != 0;}
01166
01173     bool isLost() const {return state == NIT
E_HAND_STATE_LOST;}
01174
01180     bool isTracking() const {return (state &
NITE_HAND_STATE_TRACKED) != 0;}
01181
01187     bool isTouchingFov() const {return (stat
e & NITE_HAND_STATE_TOUCHING_FOV) != 0;}
01188 }
01189
01198 class HandTrackerFrameRef
01199 {
01200 public:
01206     HandTrackerFrameRef() : m_pFrame(NULL),
m_handTracker(NULL)
01207     {}
01208
01212     ~HandTrackerFrameRef()
01213     {
01214         release();
01215     }
01216
01223     HandTrackerFrameRef(const HandTrackerFra
meRef& other) : m_pFrame(NULL)
01224     {
01225         *this = other;
01226     }
01227
01233     HandTrackerFrameRef& operator=(const Han
```

```
dTrackerFrameRef& other)
01234      {
01235          setReference(other.m_handTracker, ot
01236          her.m_pFrame);
01237          niteHandTrackerFrameAddRef(m_handTra
01238          cker, m_pFrame);
01239      }
01240
01241      bool isValid() const
01242      {
01243          return m_pFrame != NULL;
01244      }
01245
01246      void release()
01247      {
01248          if (m_pFrame != NULL)
01249          {
01250              niteHandTrackerFrameRelease(m_ha
01251              ndTracker, m_pFrame);
01252          }
01253          m_pFrame = NULL;
01254          m_handTracker = NULL;
01255      }
01256
01257      const Array<HandData>& getHands() const {
01258          return m_hands;
01259      }
01260
01261      const Array<GestureData>& getGestures()
01262      const {return m_gestures;}
01263
01264      openni::VideoFrameRef getDepthFrame() co
01265      nst
01266      {
01267          return m_depthFrame;
01268      }
01269
```

```
01291
01297     uint64_t getTimestamp() const {return m_
pFrame->timestamp;}
01298
01306     int getFrameIndex() const {return m_pFra
me->frameIndex;}
01307 private:
01308     friend class HandTracker;
01309
01310     void setReference(NiteHandTrackerHandle
handTracker, NiteHandTrackerFrame* pFrame)
01311     {
01312         release();
01313         m_handTracker = handTracker;
01314         m_pFrame = pFrame;
01315         m_depthFrame._setFrame(pFrame->pDept
hFrame);
01316
01317         m_hands.setData(m_pFrame->handCount,
01318             (HandData*)m_pFrame->pHands);
01319         m_gestures.setData(m_pFrame->gesture
Count, (GestureData*)m_pFrame->pGestures);
01320     }
01321
01322     NiteHandTrackerFrame* m_pFrame;
01323     NiteHandTrackerHandle m_handTracker;
01324     openni::VideoFrameRef m_depthFrame;
01325
01326     Array<HandData> m_hands;
01327     Array<GestureData> m_gestures;
01328
01329 class HandTracker
01330 {
01331     public:
01332         class NewFrameListener
01333         {
```

```
01376     public:
01377         NewFrameListener() : m_pHandTracker(
01378             NULL)
01379     {
01380         m_handTrackerCallbacks.readyForNextFrame = newFrameCallback;
01381     }
01382     virtual void onNewFrame(HandTracker&)
01383         ) = 0;
01384     private:
01385         friend class HandTracker;
01386         NiteHandTrackerCallbacks m_handTrackerCallbacks;
01387
01388         NiteHandTrackerCallbacks& getCallbacks() {return m_handTrackerCallbacks;}
01389
01390         static void ONI_CALLBACK_TYPE newFrameCallback(void* pCookie)
01391         {
01392             NewFrameListener* pListener = (NewFrameListener*)pCookie;
01393             pListener->onNewFrame(*pListener->m_pHandTracker);
01394         }
01395
01396         void setHandTracker(HandTracker* pHan
01397 dTracker)
01398         {
01399             m_pHandTracker = pHan
01400 dTracker;
01401         }
01402         HandTracker* m_pHandTracker;
01403     };
01404
01405     HandTracker() : m_handTrackerHandle(NULL)
01406     {}
01407 }
```

```
01420
01421
01422     ~HandTracker()
01423
01424     {
01425         destroy();
01426     }
01427
01428
01429     Status create(openni::Device* pDevice =
01430 NULL)
01431     {
01432         if (isValid())
01433         {
01434             // tracker already active
01435             return STATUS_OUT_OF_FLOW;
01436         }
01437
01438         if (pDevice == NULL)
01439         {
01440             return (Status)niteInitializeHan
01441 dTracker(&m_handTrackerHandle);
01442                 // Pick a device
01443             }
01444             return (Status)niteInitializeHandTra
01445 ckerByDevice(pDevice, &m_handTrackerHandle);
01446         }
01447
01448     void destroy()
01449     {
01450         if (isValid())
01451         {
01452             niteShutdownHandTracker(m_handTr
01453 ackerHandle);
01454             m_handTrackerHandle = NULL;
01455         }
01456     }
01457
01458     Status readFrame(HandTrackerFrameRef* pF
01459 rame)
```

```
01484     {
01485         NiteHandTrackerFrame *pNiteFrame = NULL;
01486         Status rc = (Status)niteReadHandTrackerFrame(m_handTrackerHandle, &pNiteFrame);
01487         pFrame->setReference(m_handTrackerHandle, pNiteFrame);
01488
01489         return rc;
01490     }
01491
01507     bool isValid() const
01508     {
01509         return m_handTrackerHandle != NULL;
01510     }
01511
01520     Status setSmoothingFactor(float factor)
01521     {
01522         return (Status)niteSetHandSmoothingFactor(m_handTrackerHandle, factor);
01523     }
01524
01532     float getSmoothingFactor() const
01533     {
01534         float factor;
01535         Status rc = (Status)niteGetHandSmoothingFactor(m_handTrackerHandle, &factor);
01536         if (rc != STATUS_OK)
01537         {
01538             factor = 0;
01539         }
01540         return factor;
01541     }
01542
01562     Status startHandTracking(const Point3f&
position, HandId* pNewHandId)
01563     {
```

```
01564         return (Status)niteStartHandTracking
(m_handTrackerHandle, (const NitePoint3f*)&positio
n, pNewHandId);
01565     }
01566
01573     void stopHandTracking(HandId id)
01574     {
01575         niteStopHandTracking(m_handTrackerHa
ndle, id);
01576     }
01577
01586     void addNewFrameListener(NewFrameListener
* pListener)
01587     {
01588         niteRegisterHandTrackerCallbacks(m_h
andTrackerHandle, &pListener->getCallbacks(), pLi
stener);
01589         pListener->setHandTracker(this);
01590     }
01591
01600     void removeNewFrameListener(NewFrameList
ener* pListener)
01601     {
01602         niteUnregisterHandTrackerCallbacks(m_
handTrackerHandle, &pListener->getCallbacks());
01603         pListener->setHandTracker(NULL);
01604     }
01605
01632     Status startGestureDetection(GestureType
type)
01633     {
01634         return (Status)niteStartGestureDetec
tion(m_handTrackerHandle, (NiteGestureType)type);
01635     }
01636
01643     void stopGestureDetection(GestureType ty
pe)
```

```
01644     {
01645         niteStopGestureDetection(m_handTrackerHandle, (NiteGestureType)type);
01646     }
01647
01670     Status convertHandCoordinatesToDepth(float x, float y, float z, float* pOutX, float* pOutY) const
01671     {
01672         return (Status)niteConvertHandCoordinatesToDepth(m_handTrackerHandle, x, y, z, pOutX, pOutY);
01673     }
01674
01697     Status convertDepthCoordinatesToHand(int x, int y, int z, float* pOutX, float* pOutY) const
01698     {
01699         return (Status)niteConvertDepthCoordinatesToHand(m_handTrackerHandle, x, y, z, pOutX, pOutY);
01700     }
01701
01702 private:
01703     NiteHandTrackerHandle m_handTrackerHandle;
01704 };
01705
01713 class NiTE
01714 {
01715 public:
01721     static Status initialize()
01722     {
01723         return (Status)niteInitialize();
01724     }
01725
01730     static void shutdown()
```

```
01731     {
01732         niteShutdown();
01733     }
01734
01744     static Version getVersion()
01745     {
01746         NiteVersion version = niteGetVersion
01747         ();
01748         union
01749         {
01750             NiteVersion* pC;
01751             Version* pCpp;
01751         } a;
01752         a.pC = &version;
01753         return *a.pCpp;
01754     }
01755 private:
01756     NiTE();
01757 };
01758
01759 } // namespace nite
01760
01761 #endif // _NITE_H_
```



nite::Array< T > Member List

This is the complete list of members for [nite::Array< T >](#), including all inherited members.

Array()	nite::Array< T >	[inline]
getSize() const	nite::Array< T >	[inline]
isEmpty() const	nite::Array< T >	[inline]
operator[](int index) const	nite::Array< T >	[inline]
setData(int size, T *data)	nite::Array< T >	[inline]

Graph Legend

This page explains how to interpret the graphs that are generated by doxygen.

Consider the following example:

```
/*! Invisible class because of truncation */
class Invisible { };

/*! Truncated class, inheritance relation is hidden */
class Truncated : public Invisible { };

/* Class not documented with doxygen comments */
class Undocumented { };

/*! Class that is inherited using public inheritance */
class PublicBase : public Truncated { };

/*! A template class */
template<class T> class Templ { };

/*! Class that is inherited using protected inheritance */
class ProtectedBase { };

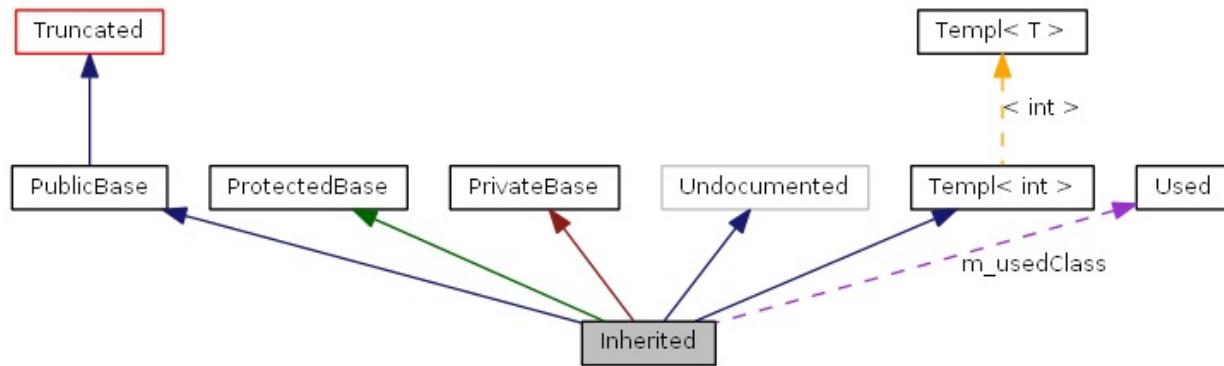
/*! Class that is inherited using private inheritance */
class PrivateBase { };
```

```

/*! Class that is used by the Inherited class */
class Used { };

/*! Super class that inherits a number of other classes */
class Inherited : public PublicBase,
                    protected ProtectedBase,
                    private PrivateBase,
                    public Undocumented,
                    public Templ<int>
{
    private:
        Used *m_usedClass;
};
```

This will result in the following graph:



The boxes in the above graph have the following meaning:

- A filled gray box represents the struct or class for which the graph is generated.
- A box with a black border denotes a documented struct or class.
- A box with a grey border denotes an undocumented struct or class.
- A box with a red border denotes a documented struct or class for which not all inheritance/containment relations are shown. A graph is truncated if it does not fit within the specified boundaries.

The arrows have the following meaning:

- A dark blue arrow is used to visualize a public inheritance relation between two classes.
- A dark green arrow is used for protected inheritance.
- A dark red arrow is used for private inheritance.
- A purple dashed arrow is used if a class is contained or used by another class. The arrow is labeled with the variable(s) through which the pointed class or struct is accessible.
- A yellow dashed arrow denotes a relation between a template instance and the template class it was instantiated from. The arrow is labeled with the template parameters of the instance.



nite::BoundingBox Member List

This is the complete list of members for [nite::BoundingBox](#), including all inherited members.

BoundingBox()	nite::BoundingBox [inline]
BoundingBox(const Point3f &min, const Point3f &max)	nite::BoundingBox [inline]
max	NiteBoundingBox
min	NiteBoundingBox



nite::GestureData Member List

This is the complete list of members for [nite::GestureData](#), including all inherited members.

[getCurrentPosition\(\) const nite::GestureData](#) [inline]

[getType\(\) const nite::GestureData](#) [inline]

[isComplete\(\) const nite::GestureData](#) [inline]

[isInProgress\(\) const nite::GestureData](#) [inline]



nite::HandData Member List

This is the complete list of members for [nite::HandData](#), including all inherited members.

getId() const	nite::HandData [inline]
getPosition() const	nite::HandData [inline]
isLost() const	nite::HandData [inline]
isNew() const	nite::HandData [inline]
isTouchingFov() const	nite::HandData [inline]
isTracking() const	nite::HandData [inline]



nite::HandTracker Member List

This is the complete list of members for [nite::HandTracker](#), including all inherited members.

addNewFrameListener (NewFrameListener *pListener)	nite::HandTracker	[in]
convertDepthCoordinatesToHand (int x, int y, int z, float *pOutX, float *pOutY) const	nite::HandTracker	[in]
convertHandCoordinatesToDepth (float x, float y, float z, float *pOutX, float *pOutY) const	nite::HandTracker	[in]
create (openni::Device *pDevice=NULL)	nite::HandTracker	[in]
destroy ()	nite::HandTracker	[in]
getSmoothingFactor () const	nite::HandTracker	[in]
HandTracker ()	nite::HandTracker	[in]
isValid () const	nite::HandTracker	[in]
readFrame (HandTrackerFrameRef *pFrame)	nite::HandTracker	[in]
removeNewFrameListener (NewFrameListener *pListener)	nite::HandTracker	[in]
setSmoothingFactor (float factor)	nite::HandTracker	[in]
startGestureDetection (GestureType type)	nite::HandTracker	[in]
startHandTracking (const Point3f &position, HandId *pNewHandId)	nite::HandTracker	[in]
stopGestureDetection (GestureType type)	nite::HandTracker	[in]
stopHandTracking (HandId id)	nite::HandTracker	[in]
~HandTracker ()	nite::HandTracker	[in]

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



nite::HandTrackerFrameRef Member List

This is the complete list of members for [nite::HandTrackerFrameRef](#), including all inherited members.

getDepthFrame() const	nite::HandTrackerFrameRef	[inline]
getFrameIndex() const	nite::HandTrackerFrameRef	[inline]
getGestures() const	nite::HandTrackerFrameRef	[inline]
getHands() const	nite::HandTrackerFrameRef	[inline]
getTimestamp() const	nite::HandTrackerFrameRef	[inline]
HandTracker class	nite::HandTrackerFrameRef	[friend]
HandTrackerFrameRef()	nite::HandTrackerFrameRef	[inline]
HandTrackerFrameRef (const HandTrackerFrameRef &other)	nite::HandTrackerFrameRef	[inline]
isValid() const	nite::HandTrackerFrameRef	[inline]
operator=(const HandTrackerFrameRef &other)	nite::HandTrackerFrameRef	[inline]
release()	nite::HandTrackerFrameRef	[inline]
~HandTrackerFrameRef()	nite::HandTrackerFrameRef	[inline]



nite::HandTracker::NewFrameListener Member List

This is the complete list of members for [nite::HandTracker::NewFrameListener](#), including all inherited members.

[HandTracker class](#)

[nite::HandTracker::NewFrameListener](#) [f]

[NewFrameListener\(\)](#)

[nite::HandTracker::NewFrameListener](#) [i]

[onNewFrame\(HandTracker &\)=0](#)

[nite::HandTracker::NewFrameListener](#) [p]



nite::UserTracker::NewFrameListener Member List

This is the complete list of members for [nite::UserTracker::NewFrameListener](#), including all inherited members.

[NewFrameListener\(\)](#)

[nite::UserTracker::NewFrameListener](#) [inl]

[onNewFrame\(UserTracker&\)=0](#)

[nite::UserTracker::NewFrameListener](#) [pure virtual]

[UserTracker class](#)

[nite::UserTracker::NewFrameListener](#) [friend]



nite::NiTE Member List

This is the complete list of members for [nite::NiTE](#), including all inherited members.

[**getVersion\(\)**](#) [nite::NiTE](#) [inline, static]

[**initialize\(\)**](#) [nite::NiTE](#) [inline, static]

[**shutdown\(\)**](#) [nite::NiTE](#) [inline, static]



NiteCTypes.h

Go to the documentation of this file.

```
00001 /*****
* ****
00002 *
*
00003 *      PrimeSense NiTE 2.0
*
00004 *      Copyright (C) 2012 PrimeSense Ltd.
*
00005 *
*
00006 *****/
00007
00008 #ifndef _NITE_C_TYPES_H_
00009 #define _NITE_C_TYPES_H_
00010
00011 #include <OniCTypes.h>
00012 #include "NiteCEnums.h"
00013
00014 typedef short int NiteUserId;
00015
00016 typedef struct NiteUserTracker* NiteUserTrackerHandle;
00017
00018 #define NITE_JOINT_COUNT 15
00019 #define NITE_POSE_COUNT 2
00020
```

```
00022 typedef struct
00023 {
00024     float x, y, z;
00025 } NitePoint3f;
00026
00028 typedef struct
00029 {
00030     float x, y, z, w;
00031 } NiteQuaternion;
00032
00034 typedef struct
00035 {
00037     NiteJointType jointType;
00038
00040     NitePoint3f position;
00041     float positionConfidence;
00042
00044     NiteQuaternion orientation;
00045     float orientationConfidence;
00046 } NiteSkeletonJoint;
00047
00049 typedef struct
00050 {
00051     NitePoint3f min;
00052     NitePoint3f max;
00053 } NiteBoundingBox;
00054
00055 typedef struct
00056 {
00057     NitePoseType type;
00058     int state;
00059 } NitePoseData;
00060
00062 typedef struct
00063 {
00064     NiteSkeletonJoint joints[NITE_JOINT_COUNT
];
```

```
00065     NiteSkeletonState state;
00066 } NiteSkeleton;
00067
00069 typedef struct
00070 {
00071     NiteUserId id;
00072     NiteBoundingBox boundingBox;
00073     NitePoint3f centerOfMass;
00074
00075     int state;
00076
00077     NiteSkeleton skeleton;
00078
00079     NitePoseData poses[NITE_POSE_COUNT];
00080 } NiteUserData;
00081
00083 typedef struct
00084 {
00085     NiteUserId* pixels;
00086
00087     int width;
00088     int height;
00089
00090     int stride;
00091 } NiteUserMap;
00092
00094 typedef struct
00095 {
00096     NitePoint3f point;
00097     NitePoint3f normal;
00098 } NitePlane;
00099
00101 typedef struct
00102 {
00104     int userCount;
00106     NiteUserData* pUser;
00107 }
```

```
00109     NiteUserMap userMap;
00110     OniFrame* pDepthFrame;
00111
00112     unsigned long long timestamp;
00113     int frameIndex;
00114
00115     float floorConfidence;
00116     NitePlane floor;
00117 } NiteUserTrackerFrame;
00118
00119
00120
00121
00122 typedef struct
00123 {
00124     OniGeneralCallback readyForNextFrame;
00125 } NiteUserTrackerCallbacks;
00126
00127
00128
00129
00130
00131 typedef struct
00132 {
00133     NiteHandId id;
00134     NitePoint3f position;
00135     int state;
00136 } NiteHandData;
00137
00138
00139 typedef struct
00140 {
00141     NiteGestureType type;
00142     NitePoint3f currentPosition;
00143     int state;
00144 } NiteGestureData;
00145
00146
00147 typedef struct
00148 {
00149     int handCount;
00150     NiteHandData* pHands;
00151
00152
00153
```

```
00155     int gestureCount;
00157     NiteGestureData* pGestures;
00158
00160     OniFrame* pDepthFrame;
00161
00162     unsigned long long timestamp;
00163     int frameIndex;
00164
00165 } NiteHandTrackerFrame;
00166
00167 typedef struct
00168 {
00169     OniGeneralCallback readyForNextFrame;
00170 } NiteHandTrackerCallbacks;
00171
00172 #define _NITE_DECLARE_VERSION(name)
00173
00174
00175 typedef struct
00176 {
00177     int major;
00178
00179     int minor;
00180
00181
```

```
00183     int maintenance;  
00184  
00186     int build;  
00187 } name;  
00188  
00189 _NITE_DECLARE_VERSION(NiteVersion);  
00190  
00191  
00192 typedef struct NiteHandTracker* NiteHandTrac  
kerHandle;  
00193  
00194 #endif // _NITE_C_TYPES_H_
```



NiteBoundingBox Member List

This is the complete list of members for [NiteBoundingBox](#), including all inherited members.

[max NiteBoundingBox](#)

[min NiteBoundingBox](#)



NitePlane Member List

This is the complete list of members for [NitePlane](#), including all inherited members.

[normal](#) [NitePlane](#)

[point](#) [NitePlane](#)



NitePoint3f Member List

This is the complete list of members for [NitePoint3f](#), including all inherited members.

[x NitePoint3f](#)

[y NitePoint3f](#)

[z NitePoint3f](#)



NiteQuaternion Member List

This is the complete list of members for [NiteQuaternion](#), including all inherited members.

w [NiteQuaternion](#)

x [NiteQuaternion](#)

y [NiteQuaternion](#)

z [NiteQuaternion](#)



nite::Plane Member List

This is the complete list of members for [nite::Plane](#), including all inherited members.

normal	NitePlane	
Plane()	nite::Plane	[inline]
Plane(const Point3f &point, const Point3f &normal)	nite::Plane	[inline]
point	NitePlane	



nite::Point3f Member List

This is the complete list of members for [nite::Point3f](#), including all inherited members.

operator!= (const Point3f &other) const	nite::Point3f [inline]
operator= (const Point3f &other)	nite::Point3f [inline]
operator== (const Point3f &other) const	nite::Point3f [inline]
Point3f()	nite::Point3f [inline]
Point3f (float x, float y, float z)	nite::Point3f [inline]
Point3f (const Point3f &other)	nite::Point3f [inline]
set (float x, float y, float z)	nite::Point3f [inline]
x	NitePoint3f
y	NitePoint3f
z	NitePoint3f



nite::PoseData Member List

This is the complete list of members for [nite::PoseData](#), including all inherited members.

[**getType\(\)**](#) const [nite::PoseData](#) [inline]

[**isEntered\(\)**](#) const [nite::PoseData](#) [inline]

[**isExited\(\)**](#) const [nite::PoseData](#) [inline]

[**isHeld\(\)**](#) const [nite::PoseData](#) [inline]



nite::Quaternion Member List

This is the complete list of members for [nite::Quaternion](#), including all inherited members.

Quaternion()	nite::Quaternion [inline]
Quaternion(float w, float x, float y, float z)	nite::Quaternion [inline]
w	NiteQuaternion
x	NiteQuaternion
y	NiteQuaternion
z	NiteQuaternion



nite::Skeleton Member List

This is the complete list of members for [nite::Skeleton](#), including all inherited members.

[**getJoint**\(JointType type\) const](#) [nite::Skeleton](#) [\[inline\]](#)

[**getState\(\)** const](#) [nite::Skeleton](#) [\[inline\]](#)



nite::SkeletonJoint Member List

This is the complete list of members for [nite::SkeletonJoint](#), including all inherited members.

getOrientation() const	nite::SkeletonJoint [inline]
getOrientationConfidence() const	nite::SkeletonJoint [inline]
getPosition() const	nite::SkeletonJoint [inline]
getPositionConfidence() const	nite::SkeletonJoint [inline]
getType() const	nite::SkeletonJoint [inline]



nite::UserData Member List

This is the complete list of members for [nite::UserData](#), including all inherited members.

getBoundingBox() const	nite::UserData	[inline]
getCenterOfMass() const	nite::UserData	[inline]
getId() const	nite::UserData	[inline]
getPose(PoseType type) const	nite::UserData	[inline]
getSkeleton() const	nite::UserData	[inline]
isLost() const	nite::UserData	[inline]
isNew() const	nite::UserData	[inline]
isVisible() const	nite::UserData	[inline]



nite::UserMap Member List

This is the complete list of members for [nite::UserMap](#), including all inherited members.

getHeight() const	nite::UserMap	[inline]
getPixels() const	nite::UserMap	[inline]
getStride() const	nite::UserMap	[inline]
getWidth() const	nite::UserMap	[inline]
UserTrackerFrameRef class	nite::UserMap	[friend]



nite::UserTracker Member List

This is the complete list of members for [nite::UserTracker](#), including all inherited members.

addNewFrameListener(NewFrameListener *pListener)	nite::UserTracker [inl.]
convertDepthCoordinatesToJoint(int x, int y, int z, float *pOutX, float *pOutY) const	nite::UserTracker [inl.]
convertJointCoordinatesToDepth(float x, float y, float z, float *pOutX, float *pOutY) const	nite::UserTracker [inl.]
create(openni::Device *pDevice=NULL)	nite::UserTracker [inl.]
destroy()	nite::UserTracker [inl.]
getSkeletonSmoothingFactor() const	nite::UserTracker [inl.]
isValid() const	nite::UserTracker [inl.]
readFrame(UserTrackerFrameRef *pFrame)	nite::UserTracker [inl.]
removeNewFrameListener(NewFrameListener *pListener)	nite::UserTracker [inl.]
setSkeletonSmoothingFactor(float factor)	nite::UserTracker [inl.]
startPoseDetection(UserId user, PoseType type)	nite::UserTracker [inl.]
startSkeletonTracking(UserId id)	nite::UserTracker [inl.]
stopPoseDetection(UserId user, PoseType type)	nite::UserTracker [inl.]
stopSkeletonTracking(UserId id)	nite::UserTracker [inl.]
UserTracker()	nite::UserTracker [inl.]
~UserTracker()	nite::UserTracker [inl.]

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1

nite::UserTrackerFrameRef Member List

This is the complete list of members for [nite::UserTrackerFrameRef](#), including all inherited members.

getDepthFrame()	nite::UserTrackerFrameRef [inline]
getFloor() const	nite::UserTrackerFrameRef [inline]
getFloorConfidence() const	nite::UserTrackerFrameRef [inline]
getFrameIndex() const	nite::UserTrackerFrameRef [inline]
getTimestamp() const	nite::UserTrackerFrameRef [inline]
getUserById(UserId id) const	nite::UserTrackerFrameRef [inline]
getUserMap() const	nite::UserTrackerFrameRef [inline]
getUsers() const	nite::UserTrackerFrameRef [inline]
isValid() const	nite::UserTrackerFrameRef [inline]
operator=(const UserTrackerFrameRef &other)	nite::UserTrackerFrameRef [inline]
release()	nite::UserTrackerFrameRef [inline]
User class	nite::UserTrackerFrameRef [friend]
UserTracker class	nite::UserTrackerFrameRef [friend]
UserTrackerFrameRef()	nite::UserTrackerFrameRef [inline]
UserTrackerFrameRef(const UserTrackerFrameRef &other)	nite::UserTrackerFrameRef [inline]
~UserTrackerFrameRef()	nite::UserTrackerFrameRef [inline]

 **NiTE 2.0**

Main Page	Namespaces	Classes	Files											
Class List	Class Index	Class Hierarchy	Class Members											
All	Functions	Variables	Related Functions											
a	b	c	d	g	h	i	n	o	p	q	r	s	u	~

- a -

- addNewFrameListener() : **nite::UserTracker** , **nite::HandTracker**
- Array() : **nite::Array< T >**

- b -

- BoundingBox() : **nite::BoundingBox**

- c -

- convertDepthCoordinatesToHand() : **nite::HandTracker**
- convertDepthCoordinatesToJoint() : **nite::UserTracker**
- convertHandCoordinatesToDepth() : **nite::HandTracker**
- convertJointCoordinatesToDepth() : **nite::UserTracker**
- create() : **nite::UserTracker** , **nite::HandTracker**

- d -

- destroy() : **nite::UserTracker** , **nite::HandTracker**

- g -

- getBoundingBox() : **nite::UserData**
- getCenterOfMass() : **nite::UserData**
- getCurrentPosition() : **nite::GestureData**
- getDepthFrame() : **nite::UserTrackerFrameRef** , **nite::HandTrackerFrameRef**

- `getFloor() : nite::UserTrackerFrameRef`
- `getFloorConfidence() : nite::UserTrackerFrameRef`
- `getFrameIndex() : nite::UserTrackerFrameRef ,
nite::HandTrackerFrameRef`
- `getGestures() : nite::HandTrackerFrameRef`
- `getHands() : nite::HandTrackerFrameRef`
- `getHeight() : nite::UserMap`
- `getId() : nite::UserData , nite::HandData`
- `getJoint() : nite::Skeleton`
- `getOrientation() : nite::SkeletonJoint`
- `getOrientationConfidence() : nite::SkeletonJoint`
- `getPixels() : nite::UserMap`
- `getPose() : nite::UserData`
- `getPosition() : nite::SkeletonJoint , nite::HandData`
- `getPositionConfidence() : nite::SkeletonJoint`
- `getSize() : nite::Array< T >`
- `getSkeleton() : nite::UserData`
- `getSkeletonSmoothingFactor() : nite::UserTracker`
- `getSmoothingFactor() : nite::HandTracker`
- `getState() : nite::Skeleton`
- `getStride() : nite::UserMap`
- `getTimestamp() : nite::UserTrackerFrameRef ,
nite::HandTrackerFrameRef`
- `getType() : nite::SkeletonJoint , nite::PoseData ,
nite::GestureData`
- `getUserById() : nite::UserTrackerFrameRef`
- `getUserMap() : nite::UserTrackerFrameRef`
- `getUsers() : nite::UserTrackerFrameRef`
- `getVersion() : nite::NiTE`
- `getWidth() : nite::UserMap`

- h -

- `HandTracker() : nite::HandTracker`
- `HandTrackerFrameRef() : nite::HandTrackerFrameRef`

- i -

- initialize() : **nite::NiTE**
- isComplete() : **nite::GestureData**
- isEmpty() : **nite::Array< T >**
- isEntered() : **nite::PoseData**
- isExited() : **nite::PoseData**
- isHeld() : **nite::PoseData**
- isInProgress() : **nite::GestureData**
- isLost() : **nite::UserData , nite::HandData**
- isNew() : **nite::HandData , nite::UserData**
- isTouchingFov() : **nite::HandData**
- isTracking() : **nite::HandData**
- isValid() : **nite::HandTrackerFrameRef , nite::UserTracker , nite::HandTracker , nite::UserTrackerFrameRef**
- isVisible() : **nite::UserData**

- **n** -

- NewFrameListener() : **nite::UserTracker::NewFrameListener , nite::HandTracker::NewFrameListener**

- **o** -

- onNewFrame() : **nite::UserTracker::NewFrameListener , nite::HandTracker::NewFrameListener**
- operator!=() : **nite::Point3f**
- operator=() : **nite::Point3f , nite::UserTrackerFrameRef , nite::HandTrackerFrameRef**
- operator==() : **nite::Point3f**
- operator[]() : **nite::Array< T >**

- **p** -

- Plane() : **nite::Plane**
- Point3f() : **nite::Point3f**

- **q** -

- Quaternion() : **nite::Quaternion**

- r -

- `readFrame()` : **nite::UserTracker** , **nite::HandTracker**
- `release()` : **nite::HandTrackerFrameRef** ,
nite::UserTrackerFrameRef
- `removeNewFrameListener()` : **nite::HandTracker** ,
nite::UserTracker

- s -

- `set()` : **nite::Point3f**
- `setData()` : **nite::Array< T >**
- `setSkeletonSmoothingFactor()` : **nite::UserTracker**
- `setSmoothingFactor()` : **nite::HandTracker**
- `shutdown()` : **nite::NiTE**
- `startGestureDetection()` : **nite::HandTracker**
- `startHandTracking()` : **nite::HandTracker**
- `startPoseDetection()` : **nite::UserTracker**
- `startSkeletonTracking()` : **nite::UserTracker**
- `stopGestureDetection()` : **nite::HandTracker**
- `stopHandTracking()` : **nite::HandTracker**
- `stopPoseDetection()` : **nite::UserTracker**
- `stopSkeletonTracking()` : **nite::UserTracker**

- u -

- `UserTracker()` : **nite::UserTracker**
- `UserTrackerFrameRef()` : **nite::UserTrackerFrameRef**

- ~ -

- `~HandTracker()` : **nite::HandTracker**
- `~HandTrackerFrameRef()` : **nite::HandTrackerFrameRef**
- `~UserTracker()` : **nite::UserTracker**
- `~UserTrackerFrameRef()` : **nite::UserTrackerFrameRef**

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Hierarchy	Class Members
All	Functions	Variables	Related Functions

- max : [NiteBoundingBox](#)
- min : [NiteBoundingBox](#)
- normal : [NitePlane](#)
- point : [NitePlane](#)
- w : [NiteQuaternion](#)
- x : [NitePoint3f](#) , [NiteQuaternion](#)
- y : [NitePoint3f](#) , [NiteQuaternion](#)
- z : [NitePoint3f](#) , [NiteQuaternion](#)



Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Hierarchy	Class Members
All	Functions	Variables	Related Functions

- HandTracker : [nite::HandTrackerFrameRef](#) , [nite::HandTracker::NewFrameListener](#)
- User : [nite::UserTrackerFrameRef](#)
- UserTracker : [nite::UserTrackerFrameRef](#) , [nite::UserTracker::NewFrameListener](#)
- UserTrackerFrameRef : [nite::UserMap](#)



NiTE 2.0

Main Page	Namespaces	Classes	Files
Namespace List	Namespace Members		
All	Typedefs	Enumerations	Enumerator

- HandId : [nite](#)
- UserId : [nite](#)

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



NiTE 2.0

Main Page	Namespaces	Classes	Files
Namespace List	Namespace Members		
All	Typedefs	Enumerations	Enumerator

- GestureType : [nite](#)
- JointType : [nite](#)
- PoseType : [nite](#)
- SkeletonState : [nite](#)
- Status : [nite](#)



NiTE 2.0

Main Page	Namespaces	Classes	Files
Namespace List	Namespace Members		
All	Typedefs	Enumerations	Enumerator
g	j	p	s

- g -

- GESTURE_CLICK : [nite](#)
- GESTURE_HAND_RAISE : [nite](#)
- GESTURE_WAVE : [nite](#)

- j -

- JOINT_HEAD : [nite](#)
- JOINT_LEFT_ELBOW : [nite](#)
- JOINT_LEFT_FOOT : [nite](#)
- JOINT_LEFT_HAND : [nite](#)
- JOINT_LEFT_HIP : [nite](#)
- JOINT_LEFT_KNEE : [nite](#)
- JOINT_LEFT_SHOULDER : [nite](#)
- JOINT_NECK : [nite](#)
- JOINT_RIGHT_ELBOW : [nite](#)
- JOINT_RIGHT_FOOT : [nite](#)
- JOINT_RIGHT_HAND : [nite](#)
- JOINT_RIGHT_HIP : [nite](#)
- JOINT_RIGHT_KNEE : [nite](#)
- JOINT_RIGHT_SHOULDER : [nite](#)
- JOINT_TORSO : [nite](#)

- p -

- POSE_CROSSED_HANDS : [nite](#)

- POSE_PSI : [nite](#)

- S -

- SKELETON_CALIBRATING : [nite](#)
- SKELETON_CALIBRATION_ERROR_HANDS : [nite](#)
- SKELETON_CALIBRATION_ERROR_HEAD : [nite](#)
- SKELETON_CALIBRATION_ERROR_LEGS : [nite](#)
- SKELETON_CALIBRATION_ERROR_NOT_IN_POSE : [nite](#)
- SKELETON_CALIBRATION_ERROR_TORSO : [nite](#)
- SKELETON_NONE : [nite](#)
- SKELETON_TRACKED : [nite](#)
- STATUS_BAD_USER_ID : [nite](#)
- STATUS_ERROR : [nite](#)
- STATUS_OK : [nite](#)
- STATUS_OUT_OF_FLOW : [nite](#)



NiteEnums.h

Go to the documentation of this file.

```
00001 /*****
* ****
00002 *
*
00003 *      PrimeSense  2.0
*
00004 *      Copyright (C) 2012 PrimeSense Ltd.
*
00005 *
*
00006 ****
**** */
00007
00008 #ifndef _NITE_ENUMS_H_
00009 #define _NITE_ENUMS_H_
00010
00012 typedef enum
00013 {
00014     JOINT_HEAD,
00015     JOINT_NECK,
00016
00017     JOINT_LEFT_SHOULDER,
00018     JOINT_RIGHT_SHOULDER,
00019     JOINT_LEFT_ELBOW,
00020     JOINT_RIGHT_ELBOW,
00021     JOINT_LEFT_HAND,
00022     JOINT_RIGHT_HAND,
```

```
00023
00024     JOINT_TORSO,
00025
00026     JOINT_LEFT_HIP,
00027     JOINT_RIGHT_HIP,
00028     JOINT_LEFT_KNEE,
00029     JOINT_RIGHT_KNEE,
00030     JOINT_LEFT FOOT,
00031     JOINT_RIGHT FOOT,
00032 } JointType;
00033
00035 typedef enum
00036 {
00038     SKELETON_NONE,
00040     SKELETON_CALIBRATING,
00042     SKELETON_TRACKED,
00043
00045     SKELETON_CALIBRATION_ERROR_NOT_IN_POSE,
00046     SKELETON_CALIBRATION_ERROR_HANDS,
00047     SKELETON_CALIBRATION_ERROR_HEAD,
00048     SKELETON_CALIBRATION_ERROR_LEGS,
00049     SKELETON_CALIBRATION_ERROR_TORSO
00050
00051 } SkeletonState;
00052
00054 typedef enum
00055 {
00056     STATUS_OK,
00057     STATUS_ERROR,
00058     STATUS_BAD_USER_ID,
00059     STATUS_OUT_OF_FLOW
00060 } Status;
00061
00062 typedef enum
00063 {
00064     POSE_PSI,
00065     POSE_CROSSED_HANDS
```

```
00066 } PoseType;
00067
00069 typedef enum
00070 {
00071     GESTURE_WAVE,
00072     GESTURE_CLICK,
00073     GESTURE_HAND_RAISE
00074 } GestureType;
00075
00076 #endif // _NITE_ENUMS_H_
```



Main Page	Namespaces	Classes	Files	
File List	File Members			
All	Typedefs	Enumerations	Enumerator	Defines

- NiteHandId : [NiteCTypes.h](#)
- NiteHandTrackerHandle : [NiteCTypes.h](#)
- NiteUserId : [NiteCTypes.h](#)
- NiteUserTrackerHandle : [NiteCTypes.h](#)

Generated on Thu Jun 6 2013 17:48:15 for NiTE 2.0 by [doxygen](#) 1.7.5.1



- GestureType : [NiteEnums.h](#)
- JointType : [NiteEnums.h](#)
- PoseType : [NiteEnums.h](#)
- SkeletonState : [NiteEnums.h](#)
- Status : [NiteEnums.h](#)



NiTE 2.0

Main Page	Namespaces	Classes	Files	
File List	File Members			
All	Typedefs	Enumerations	Enumerator	Defines
g	j	p	s	

- g -

- GESTURE_CLICK : [NiteEnums.h](#)
- GESTURE_HAND_RAISE : [NiteEnums.h](#)
- GESTURE_WAVE : [NiteEnums.h](#)

- j -

- JOINT_HEAD : [NiteEnums.h](#)
- JOINT_LEFT_ELBOW : [NiteEnums.h](#)
- JOINT_LEFT_FOOT : [NiteEnums.h](#)
- JOINT_LEFT_HAND : [NiteEnums.h](#)
- JOINT_LEFT_HIP : [NiteEnums.h](#)
- JOINT_LEFT_KNEE : [NiteEnums.h](#)
- JOINT_LEFT_SHOULDER : [NiteEnums.h](#)
- JOINT_NECK : [NiteEnums.h](#)
- JOINT_RIGHT_ELBOW : [NiteEnums.h](#)
- JOINT_RIGHT_FOOT : [NiteEnums.h](#)
- JOINT_RIGHT_HAND : [NiteEnums.h](#)
- JOINT_RIGHT_HIP : [NiteEnums.h](#)
- JOINT_RIGHT_KNEE : [NiteEnums.h](#)
- JOINT_RIGHT_SHOULDER : [NiteEnums.h](#)
- JOINT_TORSO : [NiteEnums.h](#)

- p -

- POSE_CROSSED_HANDS : [NiteEnums.h](#)

- POSE_PSI : [NiteEnums.h](#)

- S -

- SKELETON_CALIBRATING : [NiteEnums.h](#)
- SKELETON_CALIBRATION_ERROR_HANDS : [NiteEnums.h](#)
- SKELETON_CALIBRATION_ERROR_HEAD : [NiteEnums.h](#)
- SKELETON_CALIBRATION_ERROR_LEGS : [NiteEnums.h](#)
- SKELETON_CALIBRATION_ERROR_NOT_IN_POSE :
[NiteEnums.h](#)
- SKELETON_CALIBRATION_ERROR_TORSO : [NiteEnums.h](#)
- SKELETON_NONE : [NiteEnums.h](#)
- SKELETON_TRACKED : [NiteEnums.h](#)
- STATUS_BAD_USER_ID : [NiteEnums.h](#)
- STATUS_ERROR : [NiteEnums.h](#)
- STATUS_OK : [NiteEnums.h](#)
- STATUS_OUT_OF_FLOW : [NiteEnums.h](#)



- NITE_JOINT_COUNT : [NiteCTypes.h](#)
- NITE_POSE_COUNT : [NiteCTypes.h](#)