Creacion de un algoritmo para reconocer gestos dinámicos en el LeapMotion Documentation

1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Deprecated List

Member Leap::Controller::policyFlags () const

2.1.6

Member Leap::Controller::setPolicyFlags (PolicyFlag flags) const

2.1.6

Member Leap::Device::isFlipped () const

2.1.1

Member Leap::Hand::tool (int32_t id) const

2.0

Member Leap::Hand::tools () const

2.0

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

Class Documentation

5.1 Leap::Arm Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Arm:

Collaboration diagram for Leap::Arm:

Public Member Functions

- **Arm** (HandImplementation $*$)
- LEAP_EXPORT Arm ()
- LEAP_EXPORT float width () const
- LEAP_EXPORT Vector direction () const
- LEAP_EXPORT Matrix basis () const
- LEAP_EXPORT Vector elbowPosition () const
- LEAP_EXPORT Vector wristPosition () const
- LEAP_EXPORT Vector center () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT bool operator== (const Arm &) const
- LEAP_EXPORT bool operator!= (const Arm &) const
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const Arm & invalid ()

Friends

- LEAP_EXPORT friend std::ostream & operator$<<$ (std::ostream &, const Arm &)

Additional Inherited Members

5.1.1 Detailed Description

The Arm class represents the forearm.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Arm()

```
LEAP_EXPORT Leap::Arm::Arm ( )
```

Constructs an invalid Arm object.

Get valid Arm objects from a Hand object.

Since

2.0.3

5.1.3 Member Function Documentation

5.1.3.1 basis()

```
LEAP_EXPORT Matrix Leap::Arm::basis ( ) const
```

The orthonormal basis vectors for the Arm bone as a Matrix.

Basis vectors specify the orientation of a bone.

xBasis Perpendicular to the longitudinal axis of the bone; exits the arm laterally through the sides of the wrist.

yBasis or up vector Perpendicular to the longitudinal axis of the bone; exits the top and bottom of the arm. More positive in the upward direction.

zBasis Aligned with the longitudinal axis of the arm bone. More positive toward the wrist.

The bases provided for the right arm use the right-hand rule; those for the left arm use the left-hand rule. Thus, the positive direction of the x-basis is to the right for the right arm and to the left for the left arm. You can change from right-hand to left-hand rule by multiplying the z basis vector by -1.

Note that converting the basis vectors directly into a quaternion representation is not mathematically valid. If you use quaternions, create them from the derived rotation matrix not directly from the bases.

Returns

The basis of the arm bone as a matrix.

Since

2.0.3

5.1.3.2 center()

```
LEAP_EXPORT Vector Leap::Arm::center ( ) const
```

The center of the forearm.

This location represents the midpoint of the arm between the wrist position and the elbow position.

Since

2.1.0

5.1.3.3 direction()

```
LEAP_EXPORT Vector Leap::Arm::direction ( ) const
```

The normalized direction in which the arm is pointing (from elbow to wrist).

Since

2.0.3

5.1.3.4 elbowPosition()

```
LEAP_EXPORT Vector Leap::Arm::elbowPosition ( ) const
```

The position of the elbow.

If not in view, the elbow position is estimated based on typical human anatomical proportions.

Since

2.0.3

5.1.3.5 invalid()

```
static LEAP_EXPORT const Arm& Leap::Arm::invalid ( )  [static]
```

Returns an invalid Arm object.

Returns

The invalid Arm instance.

Since

2.0.3

5.1.3.6 isValid()

```
LEAP_EXPORT bool Leap::Arm::isValid ( ) const
```

Reports whether this is a valid Arm object.

Returns

True, if this Arm object contains valid tracking data.

Since

2.0.3

5.1.3.7 operator"!=()

```
LEAP_EXPORT bool Leap::Arm::operator!= (
             const Arm &  ) const
```

Compare Arm object inequality.

Two Arm objects are equal if and only if both Arm objects represent the exact same physical arm in the same frame and both Arm objects are valid.

Since

2.0.3

5.1.3.8 operator==()

```
LEAP_EXPORT bool Leap::Arm::operator== (
             const Arm &  ) const
```

Compare Arm object equality.

Two Arm objects are equal if and only if both Arm objects represent the exact same physical arm in the same frame and both Arm objects are valid.

Since

2.0.3

5.1.3.9 toString()

```
std::string Leap::Arm::toString ( ) const  [inline]
```

A string containing a brief, human readable description of the Arm object.

Returns

A description of the Arm object as a string.

Since

2.0.3

5.1.3.10 width()

```
LEAP_EXPORT float Leap::Arm::width ( ) const
```

The average width of the arm.

Since

2.0.3

5.1.3.11 wristPosition()

```
LEAP_EXPORT Vector Leap::Arm::wristPosition ( ) const
```

The position of the wrist.

Note that the wrist position is not collocated with the end of any bone in the hand. There is a gap of a few centimeters since the carpal bones are not included in the skeleton model.

Since

2.0.3

5.1.4 Friends And Related Function Documentation

5.1.4.1 operator<<

```
LEAP_EXPORT friend std::ostream& operator<< (
             std::ostream & ,
             const Arm &  )  [friend]
```

Writes a brief, human readable description of the Arm object to an output stream.

Since

2.0.3

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

- sample/include/Leap.h

5.2 Leap::Bone Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Bone:

Collaboration diagram for Leap::Bone:

Public Types

- enum Type { TYPE_METACARPAL = 0, TYPE_PROXIMAL = 1, TYPE_INTERMEDIATE = 2, TYPE_DISTAL = 3 }

Public Member Functions

- **Bone** (BoneImplementation ∗)
- LEAP_EXPORT Bone ()
- LEAP_EXPORT Vector prevJoint () const
- LEAP_EXPORT Vector nextJoint () const
- LEAP_EXPORT Vector center () const
- LEAP_EXPORT Vector direction () const
- LEAP_EXPORT float length () const
- LEAP_EXPORT float width () const
- LEAP_EXPORT Type type () const
- LEAP_EXPORT Matrix basis () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT bool operator== (const Bone &) const
- LEAP_EXPORT bool operator!= (const Bone &) const
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const Bone & invalid ()

Friends

- LEAP_EXPORT friend std::ostream & operator<< (std::ostream &, const Bone &)

Additional Inherited Members

5.2.1 Detailed Description

The Bone class represents a tracked bone.

All fingers contain 4 bones that make up the anatomy of the finger. Get valid Bone objects from a Finger object.

Bones are ordered from base to tip, indexed from 0 to 3. Additionally, the bone's Type enum may be used to index a specific bone anatomically.

The thumb does not have a base metacarpal bone and therefore contains a valid, zero length bone at that location.

Note that Bone objects can be invalid, which means that they do not contain valid tracking data and do not correspond to a physical bone. Invalid Bone objects can be the result of asking for a Bone object from an invalid finger, indexing a bone out of range, or constructing a new bone. Test for validity with the Bone::isValid() function.

Since

2.0

5.2.2 Member Enumeration Documentation

5.2.2.1 Type

```
enum Leap::Bone::Type
```

Enumerates the names of the bones.

Members of this enumeration are returned by Bone::type() to identify a Bone object.

Since

2.0

Enumerator

TYPE_METACARPAL	Bone connected to the wrist inside the palm
TYPE_PROXIMAL	Bone connecting to the palm
TYPE_INTERMEDIATE	Bone between the tip and the base
TYPE_DISTAL	Bone at the tip of the finger

5.2.3 Constructor & Destructor Documentation

5.2.3.1 Bone()

```
LEAP_EXPORT Leap::Bone::Bone ( )
```

Constructs an invalid Bone object.

Get valid Bone objects from a Finger object.

Since

2.0

5.2.4 Member Function Documentation

5.2.4.1 basis()

```
LEAP_EXPORT Matrix Leap::Bone::basis ( ) const
```

The orthonormal basis vectors for this Bone as a Matrix.

Basis vectors specify the orientation of a bone.

xBasis Perpendicular to the longitudinal axis of the bone; exits the sides of the finger.

yBasis or up vector Perpendicular to the longitudinal axis of the bone; exits the top and bottom of the finger. More positive in the upward direction.

zBasis Aligned with the longitudinal axis of the bone. More positive toward the base of the finger.

The bases provided for the right hand use the right-hand rule; those for the left hand use the left-hand rule. Thus, the positive direction of the x-basis is to the right for the right hand and to the left for the left hand. You can change from right-hand to left-hand rule by multiplying the z basis vector by -1.

You can use the basis vectors for such purposes as measuring complex finger poses and skeletal animation.

Note that converting the basis vectors directly into a quaternion representation is not mathematically valid. If you use quaternions, create them from the derived rotation matrix not directly from the bases.

Returns

The basis of the bone as a matrix.

Since

2.0

5.2.4.2 center()

```
LEAP_EXPORT Vector Leap::Bone::center ( ) const
```

The midpoint of the bone.

Returns

The midpoint in the center of the bone.

Since

2.0

5.2.4.3 direction()

```
LEAP_EXPORT Vector Leap::Bone::direction ( ) const
```

The normalized direction of the bone from base to tip.

Returns

The normalized direction of the bone from base to tip.

Since

2.0

5.2.4.4 invalid()

```
static LEAP_EXPORT const Bone& Leap::Bone::invalid ( )  [static]
```

Returns an invalid Bone object.

You can use the instance returned by this function in comparisons testing whether a given Bone instance is valid or invalid. (You can also use the Bone::isValid() function.)

Returns

The invalid Bone instance.

Since

2.0

5.2.4.5 isValid()

```
LEAP_EXPORT bool Leap::Bone::isValid ( ) const
```

Reports whether this is a valid Bone object.

Returns

True, if this Bone object contains valid tracking data.

Since

2.0

5.2.4.6 length()

```
LEAP_EXPORT float Leap::Bone::length ( ) const
```

The estimated length of the bone in millimeters.

Returns

The length of the bone in millimeters.

Since

2.0

5.2.4.7 nextJoint()

```
LEAP_EXPORT Vector Leap::Bone::nextJoint ( ) const
```

The end of the bone, closest to the finger tip.

In anatomical terms, this is the distal end of the bone.

Returns

The Vector containing the coordinates of the next joint position.

Since

2.0

5.2.4.8 operator"!=()

```
LEAP_EXPORT bool Leap::Bone::operator!= (
             const Bone &  ) const
```

Compare Bone object inequality.

Two Bone objects are equal if and only if both Bone objects represent the exact same physical bone in the same frame and both Bone objects are valid.

Since

2.0

5.2.4.9 operator==()

```
LEAP_EXPORT bool Leap::Bone::operator== (
             const Bone &  ) const
```

Compare Bone object equality.

Two Bone objects are equal if and only if both Bone objects represent the exact same physical bone in the same frame and both Bone objects are valid.

Since

2.0

5.2.4.10 prevJoint()

```
LEAP_EXPORT Vector Leap::Bone::prevJoint ( ) const
```

The base of the bone, closest to the wrist.

In anatomical terms, this is the proximal end of the bone.

Returns

The Vector containing the coordinates of the previous joint position.

Since

2.0

5.2.4.11 toString()

```
std::string Leap::Bone::toString ( ) const  [inline]
```

A string containing a brief, human readable description of the Bone object.

Returns

A description of the Bone object as a string.

Since

2.0

5.2.4.12 type()

```
LEAP_EXPORT Type Leap::Bone::type ( ) const
```

The name of this bone.

Returns

The anatomical type of this bone as a member of the Bone::Type enumeration.

Since

2.0

5.2.4.13 width()

```
LEAP_EXPORT float Leap::Bone::width ( ) const
```

The average width of the flesh around the bone in millimeters.

Returns

The width of the flesh around the bone in millimeters.

Since

2.0

5.2.5 Friends And Related Function Documentation

5.2.5.1 operator<<

```
LEAP_EXPORT friend std::ostream& operator<< (
             std::ostream & ,
             const Bone &  )  [friend]
```

Writes a brief, human readable description of the Bone object to an output stream.

Since

2.0

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

- sample/include/Leap.h

5.3 Leap::BugReport Class Reference

Inheritance diagram for Leap::BugReport:

Collaboration diagram for Leap::BugReport:

Public Member Functions

- **BugReport** (BugReportImplementation ∗)
- LEAP_EXPORT bool **beginRecording** ()
- LEAP_EXPORT void **endRecording** ()
- LEAP_EXPORT bool **isActive** () const
- LEAP_EXPORT float **progress** () const
- LEAP_EXPORT float **duration** () const

Additional Inherited Members

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

- sample/include/Leap.h

5.4 Leap::CircleGesture Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::CircleGesture:

Collaboration diagram for Leap::CircleGesture:

Public Member Functions

- LEAP_EXPORT CircleGesture ()
- LEAP_EXPORT CircleGesture (const Gesture &rhs)
- LEAP_EXPORT Vector center () const
- LEAP_EXPORT Vector normal () const
- LEAP_EXPORT float progress () const
- LEAP_EXPORT float radius () const
- LEAP_EXPORT Pointable pointable () const

Static Public Member Functions

- static Type classType ()

Additional Inherited Members

5.4.1 Detailed Description

The CircleGesture classes represents a circular finger movement.

A circle movement is recognized when the tip of a finger draws a circle within the Leap Motion Controller field of view.

Important: To use circle gestures in your application, you must enable recognition of the circle gesture. You can enable recognition with:

Circle gestures are continuous. The CircleGesture objects for the gesture have three possible states:

State::STATE_START – The circle gesture has just started. The movement has progressed far enough for the recognizer to classify it as a circle.

State::STATE_UPDATE – The circle gesture is continuing.

State::STATE_STOP – The circle gesture is finished.

You can set the minimum radius and minimum arc length required for a movement to be recognized as a circle using the config attribute of a connected Controller object. Use the following keys to configure circle recognition:

==================================== ========== ============= ======= Key string Value type Default value Units ==================================== ========== ============= ======= Gesture.Circle.MinRadius float 5.0 mm Gesture.Circle.MinArc float 1.5 $*$ pi radians ====================================
========== ============= =======

The following example demonstrates how to set the circle configuration parameters:

The Controller object must be connected to the Leap Motion service/daemon before setting the configuration parameters.

Since

1.0

5.4.2 Constructor & Destructor Documentation

5.4.2.1 CircleGesture() [1/2]

```
LEAP_EXPORT Leap::CircleGesture::CircleGesture ( )
```

Constructs a new CircleGesture object.

An uninitialized CircleGesture object is considered invalid. Get valid instances of the CircleGesture class from a Frame object.

Since

1.0

5.4.2.2 CircleGesture() [2/2]

```
LEAP_EXPORT Leap::CircleGesture::CircleGesture (
              const Gesture & rhs )
```

Constructs a CircleGesture object from an instance of the Gesture class.

Parameters

rhs	The Gesture instance to specialize. This Gesture instance must be a CircleGesture object.

Since

1.0

5.4.3 Member Function Documentation

5.4.3.1 center()

```
LEAP_EXPORT Vector Leap::CircleGesture::center ( ) const
```

The center point of the circle within the Leap Motion frame of reference.

Returns

Vector The center of the circle in mm from the Leap Motion origin.

Since

1.0

5.4.3.2 classType()

```
static Type Leap::CircleGesture::classType ( )  [inline], [static]
```

The circle gesture type.

Returns

Type The type value designating a circle gesture.

Since

1.0

5.4.3.3 normal()

```
LEAP_EXPORT Vector Leap::CircleGesture::normal ( ) const
```

Returns the normal vector for the circle being traced.

If you draw the circle clockwise, the normal vector points in the same general direction as the pointable object drawing the circle. If you draw the circle counterclockwise, the normal points back toward the pointable. If the angle between the normal and the pointable object drawing the circle is less than 90 degrees, then the circle is clockwise.

Returns

Vector the normal vector for the circle being traced

Since

1.0

5.4.3.4 pointable()

```
LEAP_EXPORT Pointable Leap::CircleGesture::pointable ( ) const
```

The finger performing the circle gesture.

Returns

Pointable A Pointable object representing the circling finger.

Since

1.0

5.4.3.5 progress()

```
LEAP_EXPORT float Leap::CircleGesture::progress ( ) const
```

The number of times the finger tip has traversed the circle.

Progress is reported as a positive number of the number. For example, a progress value of .5 indicates that the finger has gone halfway around, while a value of 3 indicates that the finger has gone around the the circle three times.

Progress starts where the circle gesture began. Since the circle must be partially formed before the Leap Motion software can recognize it, progress will be greater than zero when a circle gesture first appears in the frame.

Returns

float A positive number indicating the gesture progress.

Since

1.0

5.4.3.6 radius()

```
LEAP_EXPORT float Leap::CircleGesture::radius ( ) const
```

The radius of the circle.

Returns

The circle radius in mm.

Since

1.0

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

- sample/include/Leap.h

5.5 Leap::Config Class Reference

`#include <Leap.h>`

Inheritance diagram for Leap::Config:

Collaboration diagram for Leap::Config:

Public Types

- enum ValueType {
  TYPE_UNKNOWN = 0, TYPE_BOOLEAN = 1, TYPE_INT32 = 2, TYPE_FLOAT = 6,
  TYPE_STRING = 8 }

Public Member Functions

- LEAP_EXPORT Config ()
- ValueType type (const std::string &key) const
- bool getBool (const std::string &key) const
- bool setBool (const std::string &key, bool value)
- int32_t getInt32 (const std::string &key) const
- bool setInt32 (const std::string &key, int32_t value)
- float getFloat (const std::string &key) const
- bool setFloat (const std::string &key, float value)
- std::string getString (const std::string &key) const
- bool setString (const std::string &key, const std::string &value)
- LEAP_EXPORT bool save ()

Additional Inherited Members

5.5.1 Detailed Description

The Config class provides access to Leap Motion system configuration information.

You can get and set gesture configuration parameters using the Config object obtained from a connected Controller object. The key strings required to identify a configuration parameter include:

==================================== ========== ============= ======= Key string Value type Default value Units ==================================== ========== ============= ======= Gesture.Circle.MinRadius float 5.0 mm Gesture.Circle.MinArc float 1.5 ∗ pi radians Gesture.Swipe.MinLength float 150 mm Gesture.Swipe.MinVelocity float 1000 mm/s Gesture.KeyTap.MinDownVelocity float 50 mm/s Gesture.↩ KeyTap.HistorySeconds float 0.1 s Gesture.KeyTap.MinDistance float 3.0 mm Gesture.ScreenTap.MinForward↩ Velocity float 50 mm/s Gesture.ScreenTap.HistorySeconds float 0.1 s Gesture.ScreenTap.MinDistance float 5.0 mm ==================================== ========== ============= =======

After setting a configuration value, you must call the Config::save() method to commit the changes. You can save after the Controller has connected to the Leap Motion service/daemon. In other words, after the Controller has dispatched the serviceConnected or connected events or Controller::isConnected is true. The configuration value changes are not persistent; your application needs to set the values every time it runs.

See also

CircleGesture
KeyTapGesture
ScreenTapGesture
SwipeGesture

Since

1.0

5.5.2 Member Enumeration Documentation

5.5.2.1 ValueType

```
enum Leap::Config::ValueType
```

Enumerates the possible data types for configuration values.

The Config::type() function returns an item from the ValueType enumeration.

Since

1.0

Enumerator

TYPE_UNKNOWN	The data type is unknown. **Since** 1.0
TYPE_BOOLEAN	A boolean value. **Since** 1.0
TYPE_INT32	A 32-bit integer. **Since** 1.0
TYPE_FLOAT	A floating-point number. **Since** 1.0
TYPE_STRING	A string of characters. **Since** 1.0

5.5.3 Constructor & Destructor Documentation

5.5.3.1 Config()

```
LEAP_EXPORT Leap::Config::Config ( )
```

Constructs a Config object. Do not create your own Config objects. Get a Config object using the Controller::config() function.

Since

1.0

5.5.4 Member Function Documentation

5.5.4.1 getBool()

```
bool Leap::Config::getBool (
             const std::string & key ) const  [inline]
```

Gets the boolean representation for the specified key.

Since

1.0

5.5.4.2 getFloat()

```
float Leap::Config::getFloat (
             const std::string & key ) const  [inline]
```

Gets the floating point representation for the specified key.

Since

1.0

5.5.4.3 getInt32()

```
int32_t Leap::Config::getInt32 (
             const std::string & key ) const  [inline]
```

Gets the 32-bit integer representation for the specified key.

Since

1.0

5.5.4.4 getString()

```
std::string Leap::Config::getString (
             const std::string & key ) const  [inline]
```

Gets the string representation for the specified key.

Since

1.0

5.5.4.5 save()

```
LEAP_EXPORT bool Leap::Config::save ( )
```

Saves the current state of the config.

Call `save()` after making a set of configuration changes. The `save()` function transfers the configuration changes to the Leap Motion service. You can save after the Controller has connected to the Leap Motion service/daemon. In other words, after the Controller has dispatched the serviceConnected or connected events or Controller::isConnected is true. The configuration value changes are not persistent; your application must set the values every time it runs.

Returns

true on success, false on failure.

Since

1.0

5.5.4.6 setBool()

```
bool Leap::Config::setBool (
             const std::string & key,
             bool value )  [inline]
```

Sets the boolean representation for the specified key.

Returns

true on success, false on failure.

Since

1.0

5.5.4.7 setFloat()

```
bool Leap::Config::setFloat (
             const std::string & key,
             float value )  [inline]
```

Sets the floating point representation for the specified key.

Returns

true on success, false on failure.

Since

1.0

5.5.4.8 setInt32()

```
bool Leap::Config::setInt32 (
             const std::string & key,
             int32_t value )  [inline]
```

Sets the 32-bit integer representation for the specified key.

Returns

true on success, false on failure.

Since

1.0

5.5.4.9 setString()

```
bool Leap::Config::setString (
             const std::string & key,
             const std::string & value )  [inline]
```

Sets the string representation for the specified key.

Returns

true on success, false on failure.

Since

1.0

5.5.4.10 type()

```
ValueType Leap::Config::type (
             const std::string & key ) const  [inline]
```

Reports the natural data type for the value related to the specified key.

Parameters

key	The key for the looking up the value in the configuration dictionary.

Returns

The native data type of the value, that is, the type that does not require a data conversion.

Since

1.0

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

- sample/include/Leap.h

5.6 Leap::ConstListIterator< L, T > Class Template Reference

Public Types

- typedef std::ptrdiff_t **difference_type**
- typedef T **value_type**
- typedef const T ∗ **pointer**
- typedef const T & **reference**
- typedef std::forward_iterator_tag **iterator_category**

Public Member Functions

- **ConstListIterator** (const L &list, int index)
- const T **operator∗** () const
- const ConstListIterator< L, T > **operator++** (int)
- const ConstListIterator< L, T > & **operator++** ()
- bool **operator!=** (const ConstListIterator< L, T > &rhs) const
- bool **operator==** (const ConstListIterator< L, T > &rhs) const

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

- sample/include/Leap.h

5.7 Leap::Controller Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Controller:

Collaboration diagram for Leap::Controller:

Public Types

- enum PolicyFlag { POLICY_DEFAULT = 0, POLICY_BACKGROUND_FRAMES = (1 << 0), POLICY_IM←
  AGES = (1 << 1), POLICY_OPTIMIZE_HMD = (1 << 2) }

Public Member Functions

- **Controller** (ControllerImplementation ∗)
- LEAP_EXPORT Controller ()
- LEAP_EXPORT Controller (Listener &listener)
- LEAP_EXPORT bool isConnected () const
- LEAP_EXPORT bool isServiceConnected () const
- LEAP_EXPORT bool hasFocus () const
- LEAP_EXPORT PolicyFlag policyFlags () const
- LEAP_EXPORT void setPolicyFlags (PolicyFlag flags) const
- LEAP_EXPORT void setPolicy (PolicyFlag policy) const
- LEAP_EXPORT void clearPolicy (PolicyFlag policy) const
- LEAP_EXPORT bool isPolicySet (PolicyFlag policy) const
- LEAP_EXPORT bool addListener (Listener &listener)
- LEAP_EXPORT bool removeListener (Listener &listener)
- LEAP_EXPORT Frame frame (int history=0) const
- LEAP_EXPORT ImageList images () const
- LEAP_EXPORT Config config () const
- LEAP_EXPORT DeviceList devices () const
- LEAP_EXPORT ScreenList **locatedScreens** () const
- LEAP_EXPORT BugReport **bugReport** () const
- LEAP_EXPORT void enableGesture (Gesture::Type type, bool enable=true) const
- LEAP_EXPORT bool isGestureEnabled (Gesture::Type type) const
- LEAP_EXPORT TrackedQuad trackedQuad () const
- LEAP_EXPORT int64_t now () const

Additional Inherited Members

5.7.1 Detailed Description

The Controller class is your main interface to the Leap Motion Controller.

Create an instance of this Controller class to access frames of tracking data and configuration information. Frame data can be polled at any time using the Controller::frame() function. Call frame() or frame(0) to get the most recent frame. Set the history parameter to a positive integer to access previous frames. A controller stores up to 60 frames in its frame history.

Polling is an appropriate strategy for applications which already have an intrinsic update loop, such as a game. You can also add an instance of a subclass of Leap::Listener to the controller to handle events as they occur. The Controller dispatches events to the listener upon initialization and exiting, on connection changes, when the application gains and loses the OS input focus, and when a new frame of tracking data is available. When these events occur, the controller object invokes the appropriate callback function defined in your subclass of Listener.

To access frames of tracking data as they become available:

1. Implement a subclass of the Listener class and override the Listener::onFrame() function.
2. In your Listener::onFrame() function, call the Controller::frame() function to access the newest frame of tracking data.
3. To start receiving frames, create a Controller object and add an instance of the Listener subclass to the Controller::addListener() function.

When an instance of a Listener subclass is added to a Controller object, it calls the Listener::onInit() function when the listener is ready for use. When a connection is established between the controller and the Leap Motion software, the controller calls the Listener::onConnect() function. At this point, your application will start receiving frames of data. The controller calls the Listener::onFrame() function each time a new frame is available. If the controller loses its connection with the Leap Motion software or device for any reason, it calls the Listener::onDisconnect() function. If the listener is removed from the controller or the controller is destroyed, it calls the Listener::onExit() function. At that point, unless the listener is added to another controller again, it will no longer receive frames of tracking data.

The Controller object is multithreaded and calls the Listener functions on its own thread, not on an application thread.

Since

1.0

5.7.2 Member Enumeration Documentation

5.7.2.1 PolicyFlag

```
enum Leap::Controller::PolicyFlag
```

The supported controller policies.

The supported policy flags are:

POLICY_BACKGROUND_FRAMES – requests that your application receives frames when it is not the foreground application for user input.

The background frames policy determines whether an application receives frames of tracking data while in the background. By default, the Leap Motion software only sends tracking data to the foreground application. Only applications that need this ability should request the background frames policy. The "Allow Background Apps" checkbox must be enabled in the Leap Motion Control Panel or this policy will be denied.

POLICY_IMAGES – request that your application receives images from the device cameras. The "Allow Images" checkbox must be enabled in the Leap Motion Control Panel or this policy will be denied.

The images policy determines whether an application receives image data from the Leap Motion sensors which each frame of data. By default, this data is not sent. Only applications that use the image data should request this policy.

POLICY_OPTIMIZE_HMD – request that the tracking be optimized for head-mounted tracking.

The optimize HMD policy improves tracking in situations where the Leap Motion hardware is attached to a head-mounted display. This policy is not granted for devices that cannot be mounted to an HMD, such as Leap Motion controllers embedded in a laptop or keyboard.

Some policies can be denied if the user has disabled the feature on their Leap Motion control panel.

Since

1.0

Enumerator

POLICY_DEFAULT	The default policy. **Since** 1.0
POLICY_BACKGROUND_FRAMES	Receive background frames. **Since** 1.0
POLICY_IMAGES	Receive raw images from sensor cameras. **Since** 2.1.0
POLICY_OPTIMIZE_HMD	Optimize the tracking for head-mounted device. **Since** 2.1.2

5.7.3 Constructor & Destructor Documentation

5.7.3.1 Controller() [1/2]

```
LEAP_EXPORT Leap::Controller::Controller ( )
```

Constructs a Controller object.

When creating a Controller object, you may optionally pass in a reference to an instance of a subclass of Leap::↩
Listener. Alternatively, you may add a listener using the Controller::addListener() function.

Since

1.0

5.7.3.2 Controller() [2/2]

```
LEAP_EXPORT Leap::Controller::Controller (
            Listener & listener )
```

Constructs a Controller object.

When creating a Controller object, you may optionally pass in a reference to an instance of a subclass of Leap::↩
Listener. Alternatively, you may add a listener using the Controller::addListener() function.

Parameters

listener	An instance of Leap::Listener implementing the callback functions for the Leap Motion events you want to handle in your application.

Since

1.0

5.7.4 Member Function Documentation

5.7.4.1 addListener()

```
LEAP_EXPORT bool Leap::Controller::addListener (
            Listener & listener )
```

Adds a listener to this Controller.

The Controller dispatches Leap Motion events to each associated listener. The order in which listener callback functions are invoked is arbitrary. If you pass a listener to the Controller's constructor function, it is automatically added to the list and can be removed with the Controller::removeListener() function.

The Controller does not keep a strong reference to the Listener instance. Ensure that you maintain a reference until the listener is removed from the controller.

Parameters

listener	A subclass of Leap::Listener implementing the callback functions for the Leap Motion events you want to handle in your application.

Returns

Whether or not the listener was successfully added to the list of listeners.

Since

1.0

5.7.4.2 clearPolicy()

```
LEAP_EXPORT void Leap::Controller::clearPolicy (
            PolicyFlag policy ) const
```

Requests clearing a policy.

Policy changes are completed asynchronously and, because they are subject to user approval or system compatibility checks, may not complete successfully. Call Controller::isPolicySet() after a suitable interval to test whether the change was accepted.

Parameters

flags	A PolicyFlag value indicating the policy to request.

Since

2.1.6

5.7.4.3 config()

```
LEAP_EXPORT Config Leap::Controller::config ( ) const
```

Returns a Config object, which you can use to query the Leap Motion system for configuration information.

Returns

The Controller's Config object.

Since

1.0

5.7.4.4 devices()

```
LEAP_EXPORT DeviceList Leap::Controller::devices ( ) const
```

The list of currently attached and recognized Leap Motion controller devices.

The Device objects in the list describe information such as the range and tracking volume.

Currently, the Leap Motion Controller only allows a single active device at a time, however there may be multiple devices physically attached and listed here. Any active device(s) are guaranteed to be listed first, however order is not determined beyond that.

Returns

The list of Leap Motion controllers.

Since

1.0

5.7.4.5 enableGesture()

```
LEAP_EXPORT void Leap::Controller::enableGesture (
             Gesture::Type type,
             bool enable = true ) const
```

Enables or disables reporting of a specified gesture type.

By default, all gesture types are disabled. When disabled, gestures of the disabled type are never reported and will not appear in the frame gesture list.

As a performance optimization, only enable recognition for the types of movements that you use in your application.

Parameters

type	The type of gesture to enable or disable. Must be a member of the Gesture::Type enumeration.
enable	True, to enable the specified gesture type; False, to disable.

See also

Controller::isGestureEnabled()

Since

1.0

5.7.4.6 frame()

```
LEAP_EXPORT Frame Leap::Controller::frame (
             int history = 0 ) const
```

Returns a frame of tracking data from the Leap Motion software. Use the optional history parameter to specify which frame to retrieve. Call frame() or frame(0) to access the most recent frame; call frame(1) to access the previous frame, and so on. If you use a history value greater than the number of stored frames, then the controller returns an invalid frame.

You can call this function in your Listener implementation to get frames at the Leap Motion frame rate:

Parameters

history	The age of the frame to return, counting backwards from the most recent frame (0) into the past and up to the maximum age (59).

Returns

The specified frame; or, if no history parameter is specified, the newest frame. If a frame is not available at the specified history position, an invalid Frame is returned.

Since

1.0

5.7.4.7 hasFocus()

```
LEAP_EXPORT bool Leap::Controller::hasFocus ( ) const
```

Reports whether this application is the focused, foreground application.

By default, your application only receives tracking information from the Leap Motion controller when it has the operating system input focus. To receive tracking data when your application is in the background, the background frames policy flag must be set.

Returns

True, if application has focus; false otherwise.

See also

Controller::setPolicyFlags()

Since

1.0

5.7.4.8 images()

```
LEAP_EXPORT ImageList Leap::Controller::images ( ) const
```

The most recent set of images from the Leap Motion cameras.

Depending on timing and the current processing frame rate, the images obtained with this function can be newer than images obtained from the current frame of tracking data.

Returns

An ImageList object containing the most recent camera images.

Since

2.2.1

5.7.4.9 isConnected()

```
LEAP_EXPORT bool Leap::Controller::isConnected ( ) const
```

Reports whether this Controller is connected to the Leap Motion service and the Leap Motion hardware is plugged in.

When you first create a Controller object, isConnected() returns false. After the controller finishes initializing and connects to the Leap Motion software and if the Leap Motion hardware is plugged in, isConnected() returns true.

You can either handle the onConnect event using a Listener instance or poll the isConnected() function if you need to wait for your application to be connected to the Leap Motion software before performing some other operation.

Returns

True, if connected; false otherwise.

Since

1.0

5.7.4.10 isGestureEnabled()

```
LEAP_EXPORT bool Leap::Controller::isGestureEnabled (
            Gesture::Type type ) const
```

Reports whether the specified gesture type is enabled.

Parameters

type	The type of gesture to check; a member of the Gesture::Type enumeration.

Returns

True, if the specified type is enabled; false, otherwise.

See also

Controller::enableGesture()

Since

1.0

5.7.4.11 isPolicySet()

```
LEAP_EXPORT bool Leap::Controller::isPolicySet (
            PolicyFlag policy ) const
```

Gets the active setting for a specific policy.

Keep in mind that setting a policy flag is asynchronous, so changes are not effective immediately after calling setPolicyFlag(). In addition, a policy request can be declined by the user. You should always set the policy flags required by your application at startup and check that the policy change request was successful after an appropriate interval.

If the controller object is not connected to the Leap Motion software, then the default state for the selected policy is returned.

Parameters

flags	A PolicyFlag value indicating the policy to query.

Returns

A boolean indicating whether the specified policy has been set.

Since

2.1.6

5.7.4.12 isServiceConnected()

```
LEAP_EXPORT bool Leap::Controller::isServiceConnected ( ) const
```

Reports whether your application has a connection to the Leap Motion daemon/service. Can be true even if the Leap Motion hardware is not available.

Since

1.2

5.7.4.13 now()

```
LEAP_EXPORT int64_t Leap::Controller::now ( ) const
```

Returns a timestamp value as close as possible to the current time. Values are in microseconds, as with all the other timestamp values.

Since

2.2.7

5.7.4.14 policyFlags()

```
LEAP_EXPORT PolicyFlag Leap::Controller::policyFlags ( ) const
```

This function has been deprecated. Use isPolicySet() instead.

Deprecated 2.1.6

5.7.4.15 removeListener()

```
LEAP_EXPORT bool Leap::Controller::removeListener (
             Listener & listener )
```

Remove a listener from the list of listeners that will receive Leap Motion events. A listener must be removed if its lifetime is shorter than the controller to which it is listening.

Parameters

listener	The listener to remove.

Returns

Whether or not the listener was successfully removed from the list of listeners.

Since

1.0

5.7.4.16 setPolicy()

```
LEAP_EXPORT void Leap::Controller::setPolicy (
            PolicyFlag policy ) const
```

Requests setting a policy.

A request to change a policy is subject to user approval and a policy can be changed by the user at any time (using the Leap Motion settings dialog). The desired policy flags must be set every time an application runs.

Policy changes are completed asynchronously and, because they are subject to user approval or system compatibility checks, may not complete successfully. Call Controller::isPolicySet() after a suitable interval to test whether the change was accepted.

Parameters

policy	A PolicyFlag value indicating the policy to request.

Since

2.1.6

5.7.4.17 setPolicyFlags()

```
LEAP_EXPORT void Leap::Controller::setPolicyFlags (
            PolicyFlag flags ) const
```

This function has been deprecated. Use setPolicy() and clearPolicy() instead.

Deprecated 2.1.6

5.7.4.18 trackedQuad()

```
LEAP_EXPORT TrackedQuad Leap::Controller::trackedQuad ( ) const
```

Note: This class is an experimental API for internal use only. It may be removed without warning.

Returns information about the currently detected quad in the scene.

If no quad is being tracked, then an invalid TrackedQuad is returned.

Since

2.2.6

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

- sample/include/Leap.h

5.8 Leap::Device Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Device:

Collaboration diagram for Leap::Device:

Public Types

- enum Type { TYPE_PERIPHERAL = 1, TYPE_LAPTOP, TYPE_KEYBOARD }

Public Member Functions

- **Device** (DeviceImplementation ∗)
- LEAP_EXPORT Device ()
- LEAP_EXPORT float horizontalViewAngle () const
- LEAP_EXPORT float verticalViewAngle () const
- LEAP_EXPORT float range () const
- LEAP_EXPORT float baseline () const
- LEAP_EXPORT float distanceToBoundary (const Vector &position) const
- LEAP_EXPORT bool isEmbedded () const
- LEAP_EXPORT bool isStreaming () const
- LEAP_EXPORT bool isFlipped () const
- LEAP_EXPORT Type type () const
- std::string serialNumber () const
- LEAP_EXPORT Vector **position** () const
- LEAP_EXPORT Matrix **orientation** () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT bool operator== (const Device &) const
- LEAP_EXPORT bool operator!= (const Device &) const
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const Device & invalid ()

Friends

- LEAP_EXPORT friend std::ostream & operator<< (std::ostream &, const Device &)

Additional Inherited Members

5.8.1 Detailed Description

The Device class represents a physically connected device.

The Device class contains information related to a particular connected device such as device id, field of view relative to the device, and the position and orientation of the device in relative coordinates.

The position and orientation describe the alignment of the device relative to the user. The alignment relative to the user is only descriptive. Aligning devices to users provides consistency in the parameters that describe user interactions.

Note that Device objects can be invalid, which means that they do not contain valid device information and do not correspond to a physical device. Test for validity with the Device::isValid() function.

Since

1.0

5.8.2 Member Enumeration Documentation

5.8.2.1 Type

```
enum Leap::Device::Type
```

The available types of Leap Motion controllers.

Since

1.2

Enumerator

TYPE_PERIPHERAL	A standalone USB peripheral. The original Leap Motion controller device. **Since** 1.2
TYPE_LAPTOP	A controller embedded in a keyboard. **Since** 1.2
TYPE_KEYBOARD	A controller embedded in a laptop computer.

5.8.3 Constructor & Destructor Documentation

5.8.3.1 Device()

```
LEAP_EXPORT Leap::Device::Device ( )
```

Constructs a Device object.

An uninitialized device is considered invalid. Get valid Device objects from a DeviceList object obtained using the Controller::devices() method.

Since

1.0

5.8.4 Member Function Documentation

5.8.4.1 baseline()

```
LEAP_EXPORT float Leap::Device::baseline ( ) const
```

The distance between the center points of the stereo sensors.

The baseline value, together with the maximum resolution, influence the maximum range.

Returns

The separation distance between the center of each sensor, in mm.

Since

2.2.5

5.8.4.2 distanceToBoundary()

```
LEAP_EXPORT float Leap::Device::distanceToBoundary (
             const Vector & position ) const
```

The distance to the nearest edge of the Leap Motion controller's view volume.

The view volume is an axis-aligned, inverted pyramid centered on the device origin and extending upward to the range limit. The walls of the pyramid are described by the horizontalViewAngle and verticalViewAngle and the roof by the range. This function estimates the distance between the specified input position and the nearest wall or roof of the view volume.

Parameters

position	The point to use for the distance calculation.

Returns

The distance in millimeters from the input position to the nearest boundary.

Since

1.0

5.8.4.3 horizontalViewAngle()

```
LEAP_EXPORT float Leap::Device::horizontalViewAngle ( ) const
```

The angle of view along the x axis of this device.

The Leap Motion controller scans a region in the shape of an inverted pyramid centered at the device's center and extending upwards. The horizontalViewAngle reports the view angle along the long dimension of the device.

Returns

The horizontal angle of view in radians.

Since

1.0

5.8.4.4 invalid()

```
static LEAP_EXPORT const Device& Leap::Device::invalid ( )  [static]
```

Returns an invalid Device object.

You can use the instance returned by this function in comparisons testing whether a given Device instance is valid or invalid. (You can also use the Device::isValid() function.)

Returns

The invalid Device instance.

Since

1.0

5.8.4.5 isEmbedded()

```
LEAP_EXPORT bool Leap::Device::isEmbedded ( ) const
```

Reports whether this device is embedded in another computer or computer peripheral.

Returns

True, if this device is embedded in a laptop, keyboard, or other computer component; false, if this device is a standalone controller.

Since

1.2

5.8.4.6 isFlipped()

```
LEAP_EXPORT bool Leap::Device::isFlipped ( ) const
```

Deprecated. Always reports false.

Since

2.1

Deprecated 2.1.1

5.8.4.7 isStreaming()

```
LEAP_EXPORT bool Leap::Device::isStreaming ( ) const
```

Reports whether this device is streaming data to your application.

Currently only one controller can provide data at a time.

Since

1.2

5.8.4.8 isValid()

```
LEAP_EXPORT bool Leap::Device::isValid ( ) const
```

Reports whether this is a valid Device object.

Returns

True, if this Device object contains valid data.

Since

1.0

5.8.4.9 operator"!=()

```
LEAP_EXPORT bool Leap::Device::operator!= (
             const Device &  ) const
```

Compare Device object inequality.

Two Device objects are equal if and only if both Device objects represent the exact same Device and both Devices are valid.

Since

1.0

5.8.4.10 operator==()

```
LEAP_EXPORT bool Leap::Device::operator== (
             const Device &  ) const
```

Compare Device object equality.

Two Device objects are equal if and only if both Device objects represent the exact same Device and both Devices are valid.

Since

1.0

5.8.4.11 range()

```
LEAP_EXPORT float Leap::Device::range ( ) const
```

The maximum reliable tracking range from the center of this device.

The range reports the maximum recommended distance from the device center for which tracking is expected to be reliable. This distance is not a hard limit. Tracking may be still be functional above this distance or begin to degrade slightly before this distance depending on calibration and extreme environmental conditions.

Returns

The recommended maximum range of the device in mm.

Since

1.0

5.8.4.12 serialNumber()

```
std::string Leap::Device::serialNumber ( ) const [inline]
```

An alphanumeric serial number unique to each device.

Consumer device serial numbers consist of 2 letters followed by 11 digits.

When using multiple devices, the serial number provides an unambiguous identifier for each device.

Since

2.2.2

5.8.4.13 toString()

```
std::string Leap::Device::toString ( ) const [inline]
```

A string containing a brief, human readable description of the Device object.

Returns

A description of the Device as a string.

Since

1.0

5.8.4.14 type()

```
LEAP_EXPORT Type Leap::Device::type ( ) const
```

The device type.

Use the device type value in the (rare) circumstances that you have an application feature which relies on a particular type of device. Current types of device include the original Leap Motion peripheral, keyboard-embedded controllers, and laptop-embedded controllers.

Returns

The physical device type as a member of the DeviceType enumeration.

Since

1.2

5.8.4.15 verticalViewAngle()

```
LEAP_EXPORT float Leap::Device::verticalViewAngle ( ) const
```

The angle of view along the z axis of this device.

The Leap Motion controller scans a region in the shape of an inverted pyramid centered at the device's center and extending upwards. The verticalViewAngle reports the view angle along the short dimension of the device.

Returns

The vertical angle of view in radians.

Since

1.0

5.8.5 Friends And Related Function Documentation

5.8.5.1 operator<<

```
LEAP_EXPORT friend std::ostream& operator<< (
            std::ostream & ,
            const Device &  )  [friend]
```

Writes a brief, human readable description of the Device object.

Since

1.0

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

- sample/include/Leap.h

5.9 Leap::DeviceList Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::DeviceList:

Collaboration diagram for Leap::DeviceList:

Public Types

- typedef ConstListIterator< DeviceList, Device > const_iterator

Public Member Functions

- **DeviceList** (const ListBaseImplementation< Device > &)
- LEAP_EXPORT DeviceList ()
- LEAP_EXPORT int count () const
- LEAP_EXPORT bool isEmpty () const
- LEAP_EXPORT Device operator[ ] (int index) const
- LEAP_EXPORT DeviceList & append (const DeviceList &other)
- LEAP_EXPORT const_iterator begin () const
- LEAP_EXPORT const_iterator end () const

Additional Inherited Members

5.9.1 Detailed Description

The DeviceList class represents a list of Device objects.

Get a DeviceList object by calling Controller::devices().

Since

1.0

5.9.2 Member Typedef Documentation

5.9.2.1 const_iterator

```
typedef ConstListIterator<DeviceList, Device> Leap::DeviceList::const_iterator
```

A C++ iterator type for this DeviceList objects.

Since

1.0

5.9.3 Constructor & Destructor Documentation

5.9.3.1 DeviceList()

```
LEAP_EXPORT Leap::DeviceList::DeviceList ( )
```

Constructs an empty list of devices.

Since

1.0

5.9.4 Member Function Documentation

5.9.4.1 append()

```
LEAP_EXPORT DeviceList& Leap::DeviceList::append (
             const DeviceList & other )
```

Appends the members of the specified DeviceList to this DeviceList.

Parameters

other	A DeviceList object containing Device objects to append to the end of this DeviceList.

Since

1.0

5.9.4.2 begin()

```
LEAP_EXPORT const_iterator Leap::DeviceList::begin ( ) const
```

The C++ iterator set to the beginning of this DeviceList.

Since

1.0

5.9.4.3 count()

```
LEAP_EXPORT int Leap::DeviceList::count ( ) const
```

Returns the number of devices in this list.

Returns

The number of devices in this list.

Since

1.0

5.9.4.4 end()

```
LEAP_EXPORT const_iterator Leap::DeviceList::end ( ) const
```

The C++ iterator set to the end of this DeviceList.

Since

1.0

5.9.4.5 isEmpty()

```
LEAP_EXPORT bool Leap::DeviceList::isEmpty ( ) const
```

Reports whether the list is empty.

Returns

True, if the list has no members.

Since

1.0

5.9.4.6 operator[]()

```
LEAP_EXPORT Device Leap::DeviceList::operator[] (
             int index ) const
```

Access a list member by its position in the list.

Parameters

index	The zero-based list position index.

Returns

The Device object at the specified index.

Since

1.0

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

- sample/include/Leap.h

5.10 Leap::Finger Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Finger:

Collaboration diagram for Leap::Finger:

Public Types

- enum Joint { **JOINT_MCP** = 0, **JOINT_PIP** = 1, **JOINT_DIP** = 2, **JOINT_TIP** = 3 }
- enum Type {
  TYPE_THUMB = 0, TYPE_INDEX = 1, TYPE_MIDDLE = 2, TYPE_RING = 3,
  TYPE_PINKY = 4 }

Public Member Functions

- **Finger** (FingerImplementation ∗)
- LEAP_EXPORT Finger ()
- LEAP_EXPORT Finger (const Pointable &)
- LEAP_EXPORT Vector jointPosition (Joint jointIx) const
- LEAP_EXPORT Bone bone (Bone::Type boneIx) const
- LEAP_EXPORT Type type () const
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const Finger & invalid ()

Additional Inherited Members

5.10.1 Detailed Description

The Finger class represents a tracked finger.

Fingers are Pointable objects that the Leap Motion software has classified as a finger. Get valid Finger objects from a Frame or a Hand object.

Fingers may be permanently associated to a hand. In this case the angular order of the finger IDs will be invariant. As fingers move in and out of view it is possible for the guessed ID of a finger to be incorrect. Consequently, it may be necessary for finger IDs to be exchanged. All tracked properties, such as velocity, will remain continuous in the API. However, quantities that are derived from the API output (such as a history of positions) will be discontinuous unless they have a corresponding ID exchange.

Note that Finger objects can be invalid, which means that they do not contain valid tracking data and do not correspond to a physical finger. Invalid Finger objects can be the result of asking for a Finger object using an ID from an earlier frame when no Finger objects with that ID exist in the current frame. A Finger object created from the Finger constructor is also invalid. Test for validity with the Finger::isValid() function.

Since

1.0

5.10.2 Member Enumeration Documentation

5.10.2.1 Joint

```
enum Leap::Finger::Joint
```

Deprecated as of version 2.0

5.10.2.2 Type

```
enum Leap::Finger::Type
```

Enumerates the names of the fingers.

Members of this enumeration are returned by Finger::type() to identify a Finger object.

Since

2.0

Enumerator

TYPE_THUMB	The thumb
TYPE_INDEX	The index or fore-finger
TYPE_MIDDLE	The middle finger
TYPE_RING	The ring finger
TYPE_PINKY	The pinky or little finger

5.10.3 Constructor & Destructor Documentation

5.10.3.1 Finger() [1/2]

```
LEAP_EXPORT Leap::Finger::Finger ( )
```

Constructs a Finger object.

An uninitialized finger is considered invalid. Get valid Finger objects from a Frame or a Hand object.

Since

1.0

5.10.3.2 Finger() [2/2]

```
LEAP_EXPORT Leap::Finger::Finger (
             const Pointable & )  [explicit]
```

If the specified Pointable object represents a finger, creates a copy of it as a Finger object; otherwise, creates an invalid Finger object.

Since

1.0

5.10.4 Member Function Documentation

5.10.4.1 bone()

```
LEAP_EXPORT Bone Leap::Finger::bone (
             Bone::Type boneIx ) const
```

The bone at a given bone index on this finger.

Parameters

bone↩ Ix	An index value from the Bone::Type enumeration identifying the bone of interest.

Returns

The Bone that has the specified bone type.

Since

2.0

5.10.4.2 invalid()

```
static LEAP_EXPORT const Finger& Leap::Finger::invalid ( )  [static]
```

Returns an invalid Finger object.

You can use the instance returned by this function in comparisons testing whether a given Finger instance is valid or invalid. (You can also use the Finger::isValid() function.)

Returns

The invalid Finger instance.

Since

1.0

5.10.4.3 jointPosition()

```
LEAP_EXPORT Vector Leap::Finger::jointPosition (
             Joint jointIx ) const
```

Deprecated as of version 2.0 Use 'bone' method instead.

5.10.4.4 toString()

```
std::string Leap::Finger::toString ( ) const  [inline]
```

A string containing a brief, human readable description of the Finger object.

Returns

A description of the Finger object as a string.

Since

1.0

5.10.4.5 type()

```
LEAP_EXPORT Type Leap::Finger::type ( ) const
```

The name of this finger.

Returns

The anatomical type of this finger as a member of the Finger::Type enumeration.

Since

2.0

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

- sample/include/Leap.h

5.11 Leap::FingerList Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::FingerList:

Collaboration diagram for Leap::FingerList:

Public Types

- typedef ConstListIterator< FingerList, Finger > const_iterator

Public Member Functions

- **FingerList** (const ListBaseImplementation< Finger > &)
- LEAP_EXPORT FingerList ()
- LEAP_EXPORT int count () const
- LEAP_EXPORT bool isEmpty () const
- LEAP_EXPORT Finger operator[ ] (int index) const
- LEAP_EXPORT FingerList & append (const FingerList &other)
- LEAP_EXPORT Finger leftmost () const
- LEAP_EXPORT Finger rightmost () const
- LEAP_EXPORT Finger frontmost () const
- LEAP_EXPORT FingerList extended () const
- LEAP_EXPORT FingerList fingerType (Finger::Type type) const
- LEAP_EXPORT const_iterator begin () const
- LEAP_EXPORT const_iterator end () const

Additional Inherited Members

5.11.1 Detailed Description

The FingerList class represents a list of Finger objects.

Get a FingerList object by calling Frame::fingers().

Since

1.0

5.11.2 Member Typedef Documentation

5.11.2.1 const_iterator

```
typedef ConstListIterator<FingerList, Finger> Leap::FingerList::const_iterator
```

A C++ iterator type for FingerList objects.

Since

1.0

5.11.3 Constructor & Destructor Documentation

5.11.3.1 FingerList()

```
LEAP_EXPORT Leap::FingerList::FingerList ( )
```

Constructs an empty list of fingers.

Since

1.0

5.11.4 Member Function Documentation

5.11.4.1 append()

```
LEAP_EXPORT FingerList& Leap::FingerList::append (
            const FingerList & other )
```

Appends the members of the specified FingerList to this FingerList.

Parameters

other	A FingerList object containing Finger objects to append to the end of this FingerList.

Since

1.0

5.11.4.2 begin()

```
LEAP_EXPORT const_iterator Leap::FingerList::begin ( ) const
```

The C++ iterator set to the beginning of this FingerList.

Since

1.0

5.11.4.3 count()

```
LEAP_EXPORT int Leap::FingerList::count ( ) const
```

Returns the number of fingers in this list.

Returns

The number of fingers in this list.

Since

1.0

5.11.4.4 end()

```
LEAP_EXPORT const_iterator Leap::FingerList::end ( ) const
```

The C++ iterator set to the end of this FingerList.

Since

1.0

5.11.4.5 extended()

```
LEAP_EXPORT FingerList Leap::FingerList::extended ( ) const
```

Returns a new list containing those fingers in the current list that are extended.

Returns

The list of extended fingers from the current list.

Since

2.0

5.11.4.6 fingerType()

```
LEAP_EXPORT FingerList Leap::FingerList::fingerType (
             Finger::Type type ) const
```

Returns a list containing fingers from the current list of a given finger type by modifying the existing list.

Returns

The list of matching fingers from the current list.

Since

2.0

5.11.4.7 frontmost()

```
LEAP_EXPORT Finger Leap::FingerList::frontmost ( ) const
```

The member of the list that is farthest to the front within the standard Leap Motion frame of reference (i.e has the smallest Z coordinate).

Returns

The frontmost finger, or invalid if list is empty.

Since

1.0

5.11.4.8 isEmpty()

```
LEAP_EXPORT bool Leap::FingerList::isEmpty ( ) const
```

Reports whether the list is empty.

Returns

True, if the list has no members.

Since

1.0

5.11.4.9 leftmost()

```
LEAP_EXPORT Finger Leap::FingerList::leftmost ( ) const
```

The member of the list that is farthest to the left within the standard Leap Motion frame of reference (i.e has the smallest X coordinate).

Returns

The leftmost finger, or invalid if list is empty.

Since

1.0

5.11.4.10 operator[]()

```
LEAP_EXPORT Finger Leap::FingerList::operator[] (
             int index ) const
```

Access a list member by its position in the list.

Parameters

index	The zero-based list position index.

Returns

The Finger object at the specified index.

Since

1.0

5.11.4.11 rightmost()

```
LEAP_EXPORT Finger Leap::FingerList::rightmost ( ) const
```

The member of the list that is farthest to the right within the standard Leap Motion frame of reference (i.e has the largest X coordinate).

Returns

The rightmost finger, or invalid if list is empty.

Since

1.0

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

- sample/include/Leap.h

5.12 Leap::FloatArray Struct Reference

```
#include <LeapMath.h>
```

Public Member Functions

- float & operator[ ] (unsigned int index)
- operator float ∗ ()
- operator const float ∗ () const

Public Attributes

- float m_array [16]

5.12.1 Detailed Description

The FloatArray struct is used to allow the returning of native float arrays without requiring dynamic memory allocation. It represents a matrix with a size up to 4x4.

Since

1.0

5.12.2 Member Function Documentation

5.12.2.1 operator const float ∗()

```
Leap::FloatArray::operator const float * ( ) const  [inline]
```

Use the Float Array anywhere a const float pointer can be used.

Since

1.0

5.12.2.2 operator float ∗()

```
Leap::FloatArray::operator float * ( )  [inline]
```

Use the Float Array anywhere a float pointer can be used.

Since

1.0

5.12.2.3 operator[]()

```
float& Leap::FloatArray::operator[ ] (
            unsigned int index )  [inline]
```

Access the elements of the float array exactly like a native array.

Since

1.0

5.12.3 Member Data Documentation

5.12.3.1 m_array

```
float Leap::FloatArray::m_array[16]
```

An array containing up to 16 entries of the matrix.

Since

1.0

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

- sample/include/LeapMath.h

5.13 Leap::Frame Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Frame:

Collaboration diagram for Leap::Frame:

Public Member Functions

- **Frame** (FrameImplementation ∗)
- LEAP_EXPORT Frame ()
- LEAP_EXPORT int64_t id () const
- LEAP_EXPORT int64_t timestamp () const
- LEAP_EXPORT HandList hands () const
- LEAP_EXPORT Hand hand (int32_t id) const
- LEAP_EXPORT PointableList pointables () const
- LEAP_EXPORT Pointable pointable (int32_t id) const
- LEAP_EXPORT FingerList fingers () const
- LEAP_EXPORT Finger finger (int32_t id) const
- LEAP_EXPORT ToolList tools () const
- LEAP_EXPORT Tool tool (int32_t id) const
- LEAP_EXPORT Gesture gesture (int32_t id) const
- LEAP_EXPORT GestureList gestures () const
- LEAP_EXPORT GestureList gestures (const Frame &sinceFrame) const
- LEAP_EXPORT ImageList images () const
- LEAP_EXPORT Vector translation (const Frame &sinceFrame) const
- LEAP_EXPORT float translationProbability (const Frame &sinceFrame) const
- LEAP_EXPORT Vector rotationAxis (const Frame &sinceFrame) const
- LEAP_EXPORT float rotationAngle (const Frame &sinceFrame) const
- LEAP_EXPORT float rotationAngle (const Frame &sinceFrame, const Vector &axis) const

- LEAP_EXPORT Matrix rotationMatrix (const Frame &sinceFrame) const
- LEAP_EXPORT float rotationProbability (const Frame &sinceFrame) const
- LEAP_EXPORT float scaleFactor (const Frame &sinceFrame) const
- LEAP_EXPORT float scaleProbability (const Frame &sinceFrame) const
- LEAP_EXPORT InteractionBox interactionBox () const
- LEAP_EXPORT float currentFramesPerSecond () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT bool operator== (const Frame &) const
- LEAP_EXPORT bool operator!= (const Frame &) const
- LEAP_EXPORT TrackedQuad trackedQuad () const
- std::string serialize () const
- void deserialize (const std::string &str)
- void **serialize** (unsigned char ∗ptr) const
- int **serializeLength** () const
- void **deserialize** (const unsigned char ∗ptr, int length)
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const Frame & invalid ()

Friends

- LEAP_EXPORT friend std::ostream & operator<< (std::ostream &, const Frame &)

Additional Inherited Members

5.13.1 Detailed Description

The Frame class represents a set of hand and finger tracking data detected in a single frame.

The Leap Motion software detects hands, fingers and tools within the tracking area, reporting their positions, orientations, gestures, and motions in frames at the Leap Motion frame rate.

Access Frame objects through an instance of the Controller class:

Implement a Listener subclass to receive a callback event when a new Frame is available.

Since

1.0

5.13.2 Constructor & Destructor Documentation

5.13.2.1 Frame()

```
LEAP_EXPORT Leap::Frame::Frame ( )
```

Constructs a Frame object.

Frame instances created with this constructor are invalid. Get valid Frame objects by calling the Controller::frame() function.

The only time you should use this constructor is before deserializing serialized frame data. Call `Frame↩ ::deserialize(string)` to recreate a saved Frame.

Since

1.0

5.13.3 Member Function Documentation

5.13.3.1 currentFramesPerSecond()

```
LEAP_EXPORT float Leap::Frame::currentFramesPerSecond ( ) const
```

The instantaneous framerate.

The rate at which the Leap Motion software is providing frames of data (in frames per second). The framerate can fluctuate depending on available computing resources, activity within the device field of view, software tracking settings, and other factors.

Returns

An estimate of frames per second of the Leap Motion Controller.

Since

1.0

5.13.3.2 deserialize()

```
void Leap::Frame::deserialize (
             const std::string & str )  [inline]
```

Decodes a byte string to replace the properties of this Frame.

A Controller object must be instantiated for this function to succeed, but it does not need to be connected. To extract gestures from the deserialized frame, you must enable the appropriate gestures first.

Any existing data in the frame is destroyed. If you have references to child objects (hands, fingers, etc.), these are preserved as long as the references remain in scope.

Note: The behavior when calling functions which take another Frame object as a parameter is undefined when either frame has been deserialized. For example, calling `gestures(sinceFrame)` on a deserialized frame or with a deserialized frame as parameter (or both) does not necessarily return all gestures that occurred between the two frames. Motion functions, like `scaleFactor(startFrame)`, are more likely to return reasonable results, but could return anomalous values in some cases.

Parameters

str	A std:string object containing the serialized bytes of a frame.

Since

2.1.0

5.13.3.3 finger()

```
LEAP_EXPORT Finger Leap::Frame::finger (
            int32_t id ) const
```

The Finger object with the specified ID in this frame.

Use the Frame::finger() function to retrieve the Finger object from this frame using an ID value obtained from a previous frame. This function always returns a Finger object, but if no finger with the specified ID is present, an invalid Finger object is returned.

Note that ID values persist across frames, but only until tracking of a particular object is lost. If tracking of a finger is lost and subsequently regained, the new Finger object representing that physical finger may have a different ID than that representing the finger in an earlier frame.

Parameters

id	The ID value of a Finger object from a previous frame.

Returns

The Finger object with the matching ID if one exists in this frame; otherwise, an invalid Finger object is returned.

Since

1.0

5.13.3.4 fingers()

```
LEAP_EXPORT FingerList Leap::Frame::fingers ( ) const
```

The list of Finger objects detected in this frame, given in arbitrary order. The list can be empty if no fingers are detected.

Use PointableList::extended() to remove non-extended fingers from the list.

Returns

The FingerList containing all Finger objects detected in this frame.

Since

1.0

5.13.3.5 gesture()

```
LEAP_EXPORT Gesture Leap::Frame::gesture (
            int32_t id ) const
```

The Gesture object with the specified ID in this frame.

Use the Frame::gesture() function to return a Gesture object in this frame using an ID obtained in an earlier frame. The function always returns a Gesture object, but if there was no update for the gesture in this frame, then an invalid Gesture object is returned.

All Gesture objects representing the same recognized movement share the same ID.

Parameters

id	The ID of an Gesture object from a previous frame.

Returns

The Gesture object in the frame with the specified ID if one exists; Otherwise, an Invalid Gesture object.

Since

1.0

5.13.3.6 gestures() [1/2]

```
LEAP_EXPORT GestureList Leap::Frame::gestures ( ) const
```

The gestures recognized or continuing in this frame.

Circle and swipe gestures are updated every frame. Tap gestures only appear in the list for a single frame.

Returns

GestureList the list of gestures.

Since

1.0

5.13.3.7 gestures() [2/2]

```
LEAP_EXPORT GestureList Leap::Frame::gestures (
             const Frame & sinceFrame ) const
```

Returns a GestureList containing all gestures that have occurred since the specified frame.

Parameters

sinceFrame	An earlier Frame object. The starting frame must still be in the frame history cache, which has a default length of 60 frames.

Returns

GestureList The list of the Gesture objects that have occurred since the specified frame.

Since

1.0

5.13.3.8 hand()

```
LEAP_EXPORT Hand Leap::Frame::hand (
             int32_t id ) const
```

The Hand object with the specified ID in this frame.

Use the Frame::hand() function to retrieve the Hand object from this frame using an ID value obtained from a previous frame. This function always returns a Hand object, but if no hand with the specified ID is present, an invalid Hand object is returned.

Note that ID values persist across frames, but only until tracking of a particular object is lost. If tracking of a hand is lost and subsequently regained, the new Hand object representing that physical hand may have a different ID than that representing the physical hand in an earlier frame.

Parameters

id	The ID value of a Hand object from a previous frame.

Returns

The Hand object with the matching ID if one exists in this frame; otherwise, an invalid Hand object is returned.

Since

1.0

5.13.3.9 hands()

```
LEAP_EXPORT HandList Leap::Frame::hands ( ) const
```

The list of Hand objects detected in this frame, given in arbitrary order. The list can be empty if no hands are detected.

Returns

The HandList containing all Hand objects detected in this frame.

Since

1.0

5.13.3.10 id()

```
LEAP_EXPORT int64_t Leap::Frame::id ( ) const
```

A unique ID for this Frame.

Consecutive frames processed by the Leap Motion software have consecutive increasing values. You can use the frame ID to avoid processing the same Frame object twice:

As well as to make sure that your application processes every frame:

Returns

The frame ID.

Since

1.0

5.13.3.11 images()

```
LEAP_EXPORT ImageList Leap::Frame::images ( ) const
```

The list of images from the Leap Motion cameras.

Returns

An ImageList object containing the camera images analyzed to create this Frame.

Since

2.1

5.13.3.12 interactionBox()

```
LEAP_EXPORT InteractionBox Leap::Frame::interactionBox ( ) const
```

The current InteractionBox for the frame. See the InteractionBox class documentation for more details on how this class should be used.

Returns

The current InteractionBox object.

Since

1.0

5.13.3.13 invalid()

```
static LEAP_EXPORT const Frame& Leap::Frame::invalid ( )  [static]
```

Returns an invalid Frame object.

You can use the instance returned by this function in comparisons testing whether a given Frame instance is valid or invalid. (You can also use the Frame::isValid() function.)

Returns

The invalid Frame instance.

Since

1.0

5.13.3.14 isValid()

```
LEAP_EXPORT bool Leap::Frame::isValid ( ) const
```

Reports whether this Frame instance is valid.

A valid Frame is one generated by the Leap::Controller object that contains tracking data for all detected entities. An invalid Frame contains no actual tracking data, but you can call its functions without risk of a null pointer exception. The invalid Frame mechanism makes it more convenient to track individual data across the frame history. For example, you can invoke:

for an arbitrary Frame history value, "n", without first checking whether frame(n) returned a null object. (You should still check that the returned Finger instance is valid.)

Returns

True, if this is a valid Frame object; false otherwise.

Since

1.0

5.13.3.15 operator"!=()

```
LEAP_EXPORT bool Leap::Frame::operator!= (
             const Frame & ) const
```

Compare Frame object inequality.

Two Frame objects are equal if and only if both Frame objects represent the exact same frame of tracking data and both Frame objects are valid.

Since

1.0

5.13.3.16 operator==()

```
LEAP_EXPORT bool Leap::Frame::operator== (
             const Frame &  ) const
```

Compare Frame object equality.

Two Frame objects are equal if and only if both Frame objects represent the exact same frame of tracking data and both Frame objects are valid.

Since

1.0

5.13.3.17 pointable()

```
LEAP_EXPORT Pointable Leap::Frame::pointable (
             int32_t id ) const
```

The Pointable object with the specified ID in this frame.

Use the Frame::pointable() function to retrieve the Pointable object from this frame using an ID value obtained from a previous frame. This function always returns a Pointable object, but if no finger or tool with the specified ID is present, an invalid Pointable object is returned.

Note that ID values persist across frames, but only until tracking of a particular object is lost. If tracking of a finger or tool is lost and subsequently regained, the new Pointable object representing that finger or tool may have a different ID than that representing the finger or tool in an earlier frame.

Parameters

id	The ID value of a Pointable object from a previous frame.

Returns

The Pointable object with the matching ID if one exists in this frame; otherwise, an invalid Pointable object is returned.

Since

1.0

5.13.3.18 pointables()

```
LEAP_EXPORT PointableList Leap::Frame::pointables ( ) const
```

The list of Pointable objects (fingers and tools) detected in this frame, given in arbitrary order. The list can be empty if no fingers or tools are detected.

Use PointableList::extended() to remove non-extended fingers from the list.

Returns

The PointableList containing all Pointable objects detected in this frame.

Since

1.0

5.13.3.19 rotationAngle() [1/2]

```
LEAP_EXPORT float Leap::Frame::rotationAngle (
             const Frame & sinceFrame ) const
```

The angle of rotation around the rotation axis derived from the overall rotational motion between the current frame and the specified frame.

The returned angle is expressed in radians measured clockwise around the rotation axis (using the right-hand rule) between the start and end frames. The value is always between 0 and pi radians (0 and 180 degrees).

The Leap Motion software derives frame rotation from the relative change in position and orientation of all objects detected in the field of view.

If either this frame or sinceFrame is an invalid Frame object, then the angle of rotation is zero.

Parameters

sinceFrame	The starting frame for computing the relative rotation.

Returns

A positive value containing the heuristically determined rotational change between the current frame and that specified in the sinceFrame parameter.

Since

1.0

5.13.3.20 rotationAngle() [2/2]

```
LEAP_EXPORT float Leap::Frame::rotationAngle (
             const Frame & sinceFrame,
             const Vector & axis ) const
```

The angle of rotation around the specified axis derived from the overall rotational motion between the current frame and the specified frame.

The returned angle is expressed in radians measured clockwise around the rotation axis (using the right-hand rule) between the start and end frames. The value is always between -pi and pi radians (-180 and 180 degrees).

The Leap Motion software derives frame rotation from the relative change in position and orientation of all objects detected in the field of view.

If either this frame or sinceFrame is an invalid Frame object, then the angle of rotation is zero.

Parameters

sinceFrame	The starting frame for computing the relative rotation.
axis	The axis to measure rotation around.

Returns

A value containing the heuristically determined rotational change between the current frame and that specified in the sinceFrame parameter around the given axis.

Since

1.0

5.13.3.21 rotationAxis()

```
LEAP_EXPORT Vector Leap::Frame::rotationAxis (
             const Frame & sinceFrame ) const
```

The axis of rotation derived from the overall rotational motion between the current frame and the specified frame.

The returned direction vector is normalized.

The Leap Motion software derives frame rotation from the relative change in position and orientation of all objects detected in the field of view.

If either this frame or sinceFrame is an invalid Frame object, or if no rotation is detected between the two frames, a zero vector is returned.

Parameters

sinceFrame	The starting frame for computing the relative rotation.

Returns

A normalized direction Vector representing the axis of the heuristically determined rotational change between the current frame and that specified in the sinceFrame parameter.

Since

1.0

5.13.3.22 rotationMatrix()

```
LEAP_EXPORT Matrix Leap::Frame::rotationMatrix (
             const Frame & sinceFrame ) const
```

The transform matrix expressing the rotation derived from the overall rotational motion between the current frame and the specified frame.

The Leap Motion software derives frame rotation from the relative change in position and orientation of all objects detected in the field of view.

If either this frame or sinceFrame is an invalid Frame object, then this method returns an identity matrix.

Parameters

sinceFrame	The starting frame for computing the relative rotation.

Returns

A transformation Matrix containing the heuristically determined rotational change between the current frame and that specified in the sinceFrame parameter.

Since

1.0

5.13.3.23 rotationProbability()

```
LEAP_EXPORT float Leap::Frame::rotationProbability (
            const Frame & sinceFrame ) const
```

The estimated probability that the overall motion between the current frame and the specified frame is intended to be a rotating motion.

If either this frame or sinceFrame is an invalid Frame object, then this method returns zero.

Parameters

sinceFrame	The starting frame for computing the relative rotation.

Returns

A value between 0 and 1 representing the estimated probability that the overall motion between the current frame and the specified frame is intended to be a rotating motion.

Since

1.0

5.13.3.24 scaleFactor()

```
LEAP_EXPORT float Leap::Frame::scaleFactor (
            const Frame & sinceFrame ) const
```

The scale factor derived from the overall motion between the current frame and the specified frame.

The scale factor is always positive. A value of 1.0 indicates no scaling took place. Values between 0.0 and 1.0 indicate contraction and values greater than 1.0 indicate expansion.

The Leap Motion software derives scaling from the relative inward or outward motion of all objects detected in the field of view (independent of translation and rotation).

If either this frame or sinceFrame is an invalid Frame object, then this method returns 1.0.

Parameters

sinceFrame	The starting frame for computing the relative scaling.

Returns

A positive value representing the heuristically determined scaling change ratio between the current frame and that specified in the sinceFrame parameter.

Since

1.0

5.13.3.25 scaleProbability()

```
LEAP_EXPORT float Leap::Frame::scaleProbability (
            const Frame & sinceFrame ) const
```

The estimated probability that the overall motion between the current frame and the specified frame is intended to be a scaling motion.

If either this frame or sinceFrame is an invalid Frame object, then this method returns zero.

Parameters

sinceFrame	The starting frame for computing the relative scaling.

Returns

A value between 0 and 1 representing the estimated probability that the overall motion between the current frame and the specified frame is intended to be a scaling motion.

Since

1.0

5.13.3.26 serialize()

```
std::string Leap::Frame::serialize ( ) const  [inline]
```

Encodes this Frame object as a byte string.

Returns

The serialized string encoding the data for this frame.

Since

2.1.0

5.13.3.27 timestamp()

```
LEAP_EXPORT int64_t Leap::Frame::timestamp ( ) const
```

The frame capture time in microseconds elapsed since an arbitrary point in time in the past.

Use Controller::now() to calculate the age of the frame.

Returns

The timestamp in microseconds.

Since

1.0

5.13.3.28 tool()

```
LEAP_EXPORT Tool Leap::Frame::tool (
              int32_t id ) const
```

The Tool object with the specified ID in this frame.

Use the Frame::tool() function to retrieve the Tool object from this frame using an ID value obtained from a previous frame. This function always returns a Tool object, but if no tool with the specified ID is present, an invalid Tool object is returned.

Note that ID values persist across frames, but only until tracking of a particular object is lost. If tracking of a tool is lost and subsequently regained, the new Tool object representing that tool may have a different ID than that representing the tool in an earlier frame.

Parameters

id	The ID value of a Tool object from a previous frame.

Returns

The Tool object with the matching ID if one exists in this frame; otherwise, an invalid Tool object is returned.

Since

1.0

5.13.3.29 tools()

```
LEAP_EXPORT ToolList Leap::Frame::tools ( ) const
```

The list of Tool objects detected in this frame, given in arbitrary order. The list can be empty if no tools are detected.

Returns

The ToolList containing all Tool objects detected in this frame.

Since

1.0

5.13.3.30 toString()

```
std::string Leap::Frame::toString ( ) const  [inline]
```

A string containing a brief, human readable description of the Frame object.

Returns

A description of the Frame as a string.

Since

1.0

5.13.3.31 trackedQuad()

```
LEAP_EXPORT TrackedQuad Leap::Frame::trackedQuad ( ) const
```

Note: This class is an experimental API for internal use only. It may be removed without warning.

Returns information about the currently detected quad in the scene.

If no quad is being tracked, then an invalid TrackedQuad is returned.

Since

2.2.6

5.13.3.32 translation()

```
LEAP_EXPORT Vector Leap::Frame::translation (
             const Frame & sinceFrame ) const
```

The change of position derived from the overall linear motion between the current frame and the specified frame.

The returned translation vector provides the magnitude and direction of the movement in millimeters.

The Leap Motion software derives frame translation from the linear motion of all objects detected in the field of view.

If either this frame or sinceFrame is an invalid Frame object, then this method returns a zero vector.

Parameters

sinceFrame	The starting frame for computing the relative translation.

Returns

A Vector representing the heuristically determined change in position of all objects between the current frame and that specified in the sinceFrame parameter.

Since

1.0

5.13.3.33 translationProbability()

```
LEAP_EXPORT float Leap::Frame::translationProbability (
            const Frame & sinceFrame ) const
```

The estimated probability that the overall motion between the current frame and the specified frame is intended to be a translating motion.

If either this frame or sinceFrame is an invalid Frame object, then this method returns zero.

Parameters

sinceFrame	The starting frame for computing the translation.

Returns

A value between 0 and 1 representing the estimated probability that the overall motion between the current frame and the specified frame is intended to be a translating motion.

Since

1.0

5.13.4 Friends And Related Function Documentation

5.13.4.1 operator<<

```
LEAP_EXPORT friend std::ostream& operator<< (
            std::ostream & ,
            const Frame &  )  [friend]
```

Writes a brief, human readable description of the Frame object to an output stream.

Since

1.0

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

- sample/include/Leap.h

5.14 Leap::Gesture Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Gesture:

Collaboration diagram for Leap::Gesture:

Public Types

- enum Type {
  TYPE_INVALID = -1, TYPE_SWIPE = 1, TYPE_CIRCLE = 4, TYPE_SCREEN_TAP = 5,
  TYPE_KEY_TAP = 6 }
- enum State { STATE_INVALID = -1, STATE_START = 1, STATE_UPDATE = 2, STATE_STOP = 3 }

Public Member Functions

- **Gesture** (GestureImplementation ∗)
- LEAP_EXPORT Gesture ()
- LEAP_EXPORT Gesture (const Gesture &rhs)
- LEAP_EXPORT Type type () const
- LEAP_EXPORT State state () const
- LEAP_EXPORT int32_t id () const
- LEAP_EXPORT int64_t duration () const
- LEAP_EXPORT float durationSeconds () const
- LEAP_EXPORT Frame frame () const
- LEAP_EXPORT HandList hands () const
- LEAP_EXPORT PointableList pointables () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT bool operator== (const Gesture &rhs) const
- LEAP_EXPORT bool operator!= (const Gesture &rhs) const
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const Gesture & invalid ()

Additional Inherited Members

5.14.1 Detailed Description

The Gesture class represents a recognized movement by the user.

The Leap Motion Controller watches the activity within its field of view for certain movement patterns typical of a user gesture or command. For example, a movement from side to side with the hand can indicate a swipe gesture, while a finger poking forward can indicate a screen tap gesture.

When the Leap Motion software recognizes a gesture, it assigns an ID and adds a Gesture object to the frame gesture list. For continuous gestures, which occur over many frames, the Leap Motion software updates the gesture by adding a Gesture object having the same ID and updated properties in each subsequent frame.

Important: Recognition for each type of gesture must be enabled using the Controller::enableGesture() function; otherwise **no gestures are recognized or reported**.

Subclasses of Gesture define the properties for the specific movement patterns recognized by the Leap Motion software.

The Gesture subclasses include:

CircleGesture – A circular movement by a finger.

SwipeGesture – A straight line movement by the hand with fingers extended.

ScreenTapGesture – A forward tapping movement by a finger.

KeyTapGesture – A downward tapping movement by a finger.

Circle and swipe gestures are continuous and these objects can have a state of start, update, and stop.

The screen tap gesture is a discrete gesture. The Leap Motion software only creates a single ScreenTapGesture object for each tap and it always has a stop state.

Get valid Gesture instances from a Frame object. You can get a list of gestures with the Frame::gestures() method. You can get a list of gestures since a specified frame with the `Frame::gestures(const Frame&)` method. You can also use the `Frame::gesture()` method to find a gesture in the current frame using an ID value obtained in a previous frame.

Gesture objects can be invalid. For example, when you get a gesture by ID using `Frame::gesture()`, and there is no gesture with that ID in the current frame, then `gesture()` returns an Invalid Gesture object (rather than a null value). Always check object validity in situations where a gesture might be invalid.

The following keys can be used with the Config class to configure the gesture recognizer:

==================================== ========== ============= ======= Key string Value type Default value Units ==================================== ========== ============= ======= Gesture.Circle.MinRadius float 5.0 mm Gesture.Circle.MinArc float 1.5 $*$ pi radians Gesture.Swipe.MinLength float 150 mm Gesture.Swipe.MinVelocity float 1000 mm/s Gesture.KeyTap.MinDownVelocity float 50 mm/s Gesture.↩ KeyTap.HistorySeconds float 0.1 s Gesture.KeyTap.MinDistance float 3.0 mm Gesture.ScreenTap.MinForward↩ Velocity float 50 mm/s Gesture.ScreenTap.HistorySeconds float 0.1 s Gesture.ScreenTap.MinDistance float 5.0 mm ==================================== ========== ============= =======

Since

1.0

5.14.2 Member Enumeration Documentation

5.14.2.1 State

```
enum Leap::Gesture::State
```

The possible gesture states.

Since

1.0

Enumerator

STATE_INVALID	An invalid state **Since** 1.0
STATE_START	The gesture is starting. Just enough has happened to recognize it. **Since** 1.0
STATE_UPDATE	The gesture is in progress. (Note: not all gestures have updates). **Since** 1.0
STATE_STOP	The gesture has completed or stopped. **Since** 1.0

5.14.2.2 Type

```
enum Leap::Gesture::Type
```

The supported types of gestures.

Since

1.0

Enumerator

TYPE_INVALID	An invalid type. **Since** 1.0
TYPE_SWIPE	A straight line movement by the hand with fingers extended. **Since** 1.0
TYPE_CIRCLE	A circular movement by a finger. **Since** 1.0
TYPE_SCREEN_TAP	A forward tapping movement by a finger. **Since** 1.0
TYPE_KEY_TAP	A downward tapping movement by a finger. **Since** 1.0

5.14.3 Constructor & Destructor Documentation

5.14.3.1 Gesture() [1/2]

```
LEAP_EXPORT Leap::Gesture::Gesture ( )
```

Constructs a new Gesture object.

An uninitialized Gesture object is considered invalid. Get valid instances of the Gesture class, which will be one of the Gesture subclasses, from a Frame object.

Since

1.0

5.14.3.2 Gesture() [2/2]

```
LEAP_EXPORT Leap::Gesture::Gesture (
            const Gesture & rhs )
```

Constructs a new copy of an Gesture object.

Since

1.0

5.14.4 Member Function Documentation

5.14.4.1 duration()

```
LEAP_EXPORT int64_t Leap::Gesture::duration ( ) const
```

The elapsed duration of the recognized movement up to the frame containing this Gesture object, in microseconds.

The duration reported for the first Gesture in the sequence (with the STATE_START state) will typically be a small positive number since the movement must progress far enough for the Leap Motion software to recognize it as an intentional gesture.

Returns

int64_t the elapsed duration in microseconds.

Since

1.0

5.14.4.2 durationSeconds()

```
LEAP_EXPORT float Leap::Gesture::durationSeconds ( ) const
```

The elapsed duration in seconds.

See also

duration()

Returns

float the elapsed duration in seconds.

Since

1.0

5.14.4.3 frame()

```
LEAP_EXPORT Frame Leap::Gesture::frame ( ) const
```

The Frame containing this Gesture instance.

_

Returns

Frame The parent Frame object.

Since

1.0

5.14.4.4 hands()

```
LEAP_EXPORT HandList Leap::Gesture::hands ( ) const
```

The list of hands associated with this Gesture, if any.

If no hands are related to this gesture, the list is empty.

Returns

HandList the list of related Hand objects.

Since

1.0

5.14.4.5 id()

```
LEAP_EXPORT int32_t Leap::Gesture::id ( ) const
```

The gesture ID.

All Gesture objects belonging to the same recognized movement share the same ID value. Use the ID value with the Frame::gesture() method to find updates related to this Gesture object in subsequent frames.

Returns

int32_t the ID of this Gesture.

Since

1.0

5.14.4.6 invalid()

```
static LEAP_EXPORT const Gesture& Leap::Gesture::invalid ( )  [static]
```

Returns an invalid Gesture object.

You can use the instance returned by this function in comparisons testing whether a given Gesture instance is valid or invalid. (You can also use the Gesture::isValid() function.)

Returns

The invalid Gesture instance.

Since

1.0

5.14.4.7 isValid()

```
LEAP_EXPORT bool Leap::Gesture::isValid ( ) const
```

Reports whether this Gesture instance represents a valid Gesture.

An invalid Gesture object does not represent a snapshot of a recognized movement. Invalid Gesture objects are returned when a valid object cannot be provided. For example, when you get an gesture by ID using Frame↩ ::gesture(), and there is no gesture with that ID in the current frame, then gesture() returns an Invalid Gesture object (rather than a null value). Always check object validity in situations where an gesture might be invalid.

Returns

bool True, if this is a valid Gesture instance; false, otherwise.

Since

1.0

5.14.4.8 operator"!=()

```
LEAP_EXPORT bool Leap::Gesture::operator!= (
             const Gesture & rhs ) const
```

Compare Gesture object inequality.

Two Gestures are equal only if they represent the same snapshot of the same recognized movement.

Since

1.0

5.14.4.9 operator==()

```
LEAP_EXPORT bool Leap::Gesture::operator== (
             const Gesture & rhs ) const
```

Compare Gesture object equality.

Two Gestures are equal if they represent the same snapshot of the same recognized movement.

Since

1.0

5.14.4.10 pointables()

```
LEAP_EXPORT PointableList Leap::Gesture::pointables ( ) const
```

The list of fingers and tools associated with this Gesture, if any.

If no Pointable objects are related to this gesture, the list is empty.

Returns

PointableList the list of related Pointable objects.

Since

1.0

5.14.4.11 state()

```
LEAP_EXPORT State Leap::Gesture::state ( ) const
```

The gesture state.

Recognized movements occur over time and have a beginning, a middle, and an end. The 'state()' attribute reports where in that sequence this Gesture object falls.

Returns

Gesture::State A value from the Gesture::State enumeration.

Since

1.0

5.14.4.12 toString()

```
std::string Leap::Gesture::toString ( ) const [inline]
```

A string containing a brief, human-readable description of this Gesture.

Since

1.0

5.14.4.13 type()

```
LEAP_EXPORT Type Leap::Gesture::type ( ) const
```

The gesture type.

Returns

Gesture::Type A value from the Gesture::Type enumeration.

Since

1.0

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

- sample/include/Leap.h

5.15 Leap::GestureList Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::GestureList:

Collaboration diagram for Leap::GestureList:

Public Types

- typedef ConstListIterator< GestureList, Gesture > const_iterator

Public Member Functions

- **GestureList** (const ListBaseImplementation< Gesture > &)
- LEAP_EXPORT GestureList ()
- LEAP_EXPORT int count () const
- LEAP_EXPORT bool isEmpty () const
- LEAP_EXPORT Gesture operator[ ] (int index) const
- LEAP_EXPORT GestureList & append (const GestureList &other)
- LEAP_EXPORT const_iterator begin () const
- LEAP_EXPORT const_iterator end () const

Additional Inherited Members

5.15.1 Detailed Description

The GestureList class represents a list of Gesture objects.

Get a GestureList object from a Frame object.

Since

1.0

5.15.2 Member Typedef Documentation

5.15.2.1 const_iterator

```
typedef ConstListIterator<GestureList, Gesture> Leap::GestureList::const_iterator
```

A C++ iterator type for GestureList objects.

Since

1.0

5.15.3 Constructor & Destructor Documentation

5.15.3.1 GestureList()

```
LEAP_EXPORT Leap::GestureList::GestureList ( )
```

Constructs an empty gesture list.

Since

1.0

5.15.4 Member Function Documentation

5.15.4.1 append()

```
LEAP_EXPORT GestureList& Leap::GestureList::append (
             const GestureList & other )
```

Appends the members of the specified GestureList to this GestureList.

Parameters

other	A GestureList object containing Gesture objects to append to the end of this GestureList.

Since

1.0

5.15.4.2 begin()

```
LEAP_EXPORT const_iterator Leap::GestureList::begin ( ) const
```

The C++ iterator set to the beginning of this GestureList.

Since

1.0

5.15.4.3 count()

```
LEAP_EXPORT int Leap::GestureList::count ( ) const
```

The length of this list.

Returns

The number of gestures in this list.

Since

1.0

5.15.4.4 end()

```
LEAP_EXPORT const_iterator Leap::GestureList::end ( ) const
```

The C++ iterator set to the end of this GestureList.

Since

1.0

5.15.4.5 isEmpty()

```
LEAP_EXPORT bool Leap::GestureList::isEmpty ( ) const
```

Reports whether the list is empty.

Returns

True, if the list has no members.

Since

1.0

5.15.4.6 operator[]()

```
LEAP_EXPORT Gesture Leap::GestureList::operator[] (
              int index ) const
```

Access a list member by its position in the list.

Parameters

index	The zero-based list position index.

Returns

The Gesture object at the specified index.

Since

1.0

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

- sample/include/Leap.h

5.16 Leap::Hand Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Hand:

Collaboration diagram for Leap::Hand:

Public Member Functions

- **Hand** (HandImplementation ∗)
- LEAP_EXPORT Hand ()
- LEAP_EXPORT int32_t id () const
- LEAP_EXPORT Frame frame () const
- LEAP_EXPORT PointableList pointables () const
- LEAP_EXPORT Pointable pointable (int32_t id) const
- LEAP_EXPORT FingerList fingers () const
- LEAP_EXPORT Finger finger (int32_t id) const
- LEAP_EXPORT ToolList tools () const
- LEAP_EXPORT Tool tool (int32_t id) const
- LEAP_EXPORT Vector palmPosition () const
- LEAP_EXPORT Vector stabilizedPalmPosition () const
- LEAP_EXPORT Vector palmVelocity () const
- LEAP_EXPORT Vector palmNormal () const
- LEAP_EXPORT float palmWidth () const
- LEAP_EXPORT Vector direction () const
- LEAP_EXPORT Matrix basis () const
- LEAP_EXPORT Arm arm () const
- LEAP_EXPORT Vector wristPosition () const
- LEAP_EXPORT Vector sphereCenter () const
- LEAP_EXPORT float sphereRadius () const
- LEAP_EXPORT float pinchStrength () const
- LEAP_EXPORT float grabStrength () const
- LEAP_EXPORT Vector translation (const Frame &sinceFrame) const
- LEAP_EXPORT float translationProbability (const Frame &sinceFrame) const
- LEAP_EXPORT Vector rotationAxis (const Frame &sinceFrame) const
- LEAP_EXPORT float rotationAngle (const Frame &sinceFrame) const
- LEAP_EXPORT float rotationAngle (const Frame &sinceFrame, const Vector &axis) const
- LEAP_EXPORT Matrix rotationMatrix (const Frame &sinceFrame) const

- LEAP_EXPORT float rotationProbability (const Frame &sinceFrame) const
- LEAP_EXPORT float scaleFactor (const Frame &sinceFrame) const
- LEAP_EXPORT float scaleProbability (const Frame &sinceFrame) const
- LEAP_EXPORT float timeVisible () const
- LEAP_EXPORT float confidence () const
- LEAP_EXPORT bool isLeft () const
- LEAP_EXPORT bool isRight () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT bool operator== (const Hand &) const
- LEAP_EXPORT bool operator!= (const Hand &) const
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const Hand & invalid ()

Friends

- LEAP_EXPORT friend std::ostream & operator<< (std::ostream &, const Hand &)

Additional Inherited Members

5.16.1 Detailed Description

The Hand class reports the physical characteristics of a detected hand.

Hand tracking data includes a palm position and velocity; vectors for the palm normal and direction to the fingers; properties of a sphere fit to the hand; and lists of the attached fingers.

Get Hand objects from a Frame object:

Note that Hand objects can be invalid, which means that they do not contain valid tracking data and do not correspond to a physical entity. Invalid Hand objects can be the result of asking for a Hand object using an ID from an earlier frame when no Hand objects with that ID exist in the current frame. A Hand object created from the Hand constructor is also invalid. Test for validity with the Hand::isValid() function.

Since

1.0

5.16.2 Constructor & Destructor Documentation

5.16.2.1 Hand()

```
LEAP_EXPORT Leap::Hand::Hand ( )
```

Constructs a Hand object.

An uninitialized hand is considered invalid. Get valid Hand objects from a Frame object.

Since

1.0

5.16.3 Member Function Documentation

5.16.3.1 arm()

```
LEAP_EXPORT Arm Leap::Hand::arm ( ) const
```

The arm to which this hand is attached.

If the arm is not completely in view, Arm attributes are estimated based on the attributes of entities that are in view combined with typical human anatomy.

Returns

The Arm object for this hand.

Since

2.0.3

5.16.3.2 basis()

```
LEAP_EXPORT Matrix Leap::Hand::basis ( ) const
```

The orientation of the hand as a basis matrix.

The basis is defined as follows:

xAxis Positive in the direction of the pinky

yAxis Positive above the hand

zAxis Positive in the direction of the wrist

Note: Since the left hand is a mirror of the right hand, the basis matrix will be left-handed for left hands.

Returns

The basis of the hand as a matrix.

Since

2.0

5.16.3.3 confidence()

```
LEAP_EXPORT float Leap::Hand::confidence ( ) const
```

How confident we are with a given hand pose.

The confidence level ranges between 0.0 and 1.0 inclusive.

Since

2.0

5.16.3.4 direction()

```
LEAP_EXPORT Vector Leap::Hand::direction ( ) const
```

The direction from the palm position toward the fingers.

The direction is expressed as a unit vector pointing in the same direction as the directed line from the palm position to the fingers.

You can use the palm direction vector to compute the pitch and yaw angles of the palm with respect to the horizontal plane:

Returns

The Vector pointing from the palm position toward the fingers.

Since

1.0

5.16.3.5 finger()

```
LEAP_EXPORT Finger Leap::Hand::finger (
            int32_t id ) const
```

The Finger object with the specified ID attached to this hand.

Use the Hand::finger() function to retrieve a Finger object attached to this hand using an ID value obtained from a previous frame. This function always returns a Finger object, but if no finger with the specified ID is present, an invalid Finger object is returned.

Note that ID values persist across frames, but only until tracking of a particular object is lost. If tracking of a finger is lost and subsequently regained, the new Finger object representing that finger may have a different ID than that representing the finger in an earlier frame.

Parameters

id	The ID value of a Finger object from a previous frame.

Returns

The Finger object with the matching ID if one exists for this hand in this frame; otherwise, an invalid Finger object is returned.

Since

1.0

5.16.3.6 fingers()

```
LEAP_EXPORT FingerList Leap::Hand::fingers ( ) const
```

The list of Finger objects detected in this frame that are attached to this hand, given in order from thumb to pinky. The list cannot be empty.

Use PointableList::extended() to remove non-extended fingers from the list.

Returns

The FingerList containing all Finger objects attached to this hand.

Since

1.0

5.16.3.7 frame()

```
LEAP_EXPORT Frame Leap::Hand::frame ( ) const
```

The Frame associated with this Hand.

Returns

The associated Frame object, if available; otherwise, an invalid Frame object is returned.

Since

1.0

5.16.3.8 grabStrength()

```
LEAP_EXPORT float Leap::Hand::grabStrength ( ) const
```

The strength of a grab hand pose.

The strength is zero for an open hand, and blends to 1.0 when a grabbing hand pose is recognized.

Returns

A float value in the [0..1] range representing the holding strength of the pose.

Since

2.0

5.16.3.9 id()

```
LEAP_EXPORT int32_t Leap::Hand::id ( ) const
```

A unique ID assigned to this Hand object, whose value remains the same across consecutive frames while the tracked hand remains visible. If tracking is lost (for example, when a hand is occluded by another hand or when it is withdrawn from or reaches the edge of the Leap Motion Controller field of view), the Leap Motion software may assign a new ID when it detects the hand in a future frame.

Use the ID value with the Frame::hand() function to find this Hand object in future frames:

Returns

The ID of this hand.

Since

1.0

5.16.3.10 invalid()

```
static LEAP_EXPORT const Hand& Leap::Hand::invalid ( )  [static]
```

Returns an invalid Hand object.

You can use the instance returned by this function in comparisons testing whether a given Hand instance is valid or invalid. (You can also use the Hand::isValid() function.)

Returns

The invalid Hand instance.

Since

1.0

5.16.3.11 isLeft()

```
LEAP_EXPORT bool Leap::Hand::isLeft ( ) const
```

Identifies whether this Hand is a left hand.

Returns

True if the hand is identified as a left hand.

Since

2.0

5.16.3.12 isRight()

```
LEAP_EXPORT bool Leap::Hand::isRight ( ) const
```

Identifies whether this Hand is a right hand.

Returns

True if the hand is identified as a right hand.

Since

2.0

5.16.3.13 isValid()

```
LEAP_EXPORT bool Leap::Hand::isValid ( ) const
```

Reports whether this is a valid Hand object.

Returns

True, if this Hand object contains valid tracking data.

Since

1.0

5.16.3.14 operator"!=()

```
LEAP_EXPORT bool Leap::Hand::operator!= (
             const Hand &  ) const
```

Compare Hand object inequality.

Two Hand objects are equal if and only if both Hand objects represent the exact same physical hand in the same frame and both Hand objects are valid.

Since

1.0

5.16.3.15 operator==()

```
LEAP_EXPORT bool Leap::Hand::operator== (
             const Hand &  ) const
```

Compare Hand object equality.

Two Hand objects are equal if and only if both Hand objects represent the exact same physical hand in the same frame and both Hand objects are valid.

Since

1.0

5.16.3.16 palmNormal()

```
LEAP_EXPORT Vector Leap::Hand::palmNormal ( ) const
```

The normal vector to the palm. If your hand is flat, this vector will point downward, or "out" of the front surface of your palm.

The direction is expressed as a unit vector pointing in the same direction as the palm normal (that is, a vector orthogonal to the palm).

You can use the palm normal vector to compute the roll angle of the palm with respect to the horizontal plane:

Returns

The Vector normal to the plane formed by the palm.

Since

1.0

5.16.3.17 palmPosition()

```
LEAP_EXPORT Vector Leap::Hand::palmPosition ( ) const
```

The center position of the palm in millimeters from the Leap Motion Controller origin.

Returns

The Vector representing the coordinates of the palm position.

Since

1.0

5.16.3.18 palmVelocity()

```
LEAP_EXPORT Vector Leap::Hand::palmVelocity ( ) const
```

The rate of change of the palm position in millimeters/second.

Returns

The Vector representing the coordinates of the palm velocity.

Since

1.0

5.16.3.19 palmWidth()

```
LEAP_EXPORT float Leap::Hand::palmWidth ( ) const
```

The estimated width of the palm when the hand is in a flat position.

Returns

The width of the palm in millimeters

Since

2.0

5.16.3.20 pinchStrength()

```
LEAP_EXPORT float Leap::Hand::pinchStrength ( ) const
```

The holding strength of a pinch hand pose.

The strength is zero for an open hand, and blends to 1.0 when a pinching hand pose is recognized. Pinching can be done between the thumb and any other finger of the same hand.

Returns

A float value in the [0..1] range representing the holding strength of the pinch pose.

Since

2.0

5.16.3.21 pointable()

```
LEAP_EXPORT Pointable Leap::Hand::pointable (
            int32_t id ) const
```

The Pointable object with the specified ID associated with this hand.

Use the Hand::pointable() function to retrieve a Pointable object associated with this hand using an ID value obtained from a previous frame. This function always returns a Pointable object, but if no finger with the specified ID is present, an invalid Pointable object is returned.

Note that the ID values assigned to fingers are based on the hand ID. Hand IDs persist across frames, but only until tracking of that hand is lost. If tracking of the hand is lost and subsequently regained, the new Hand object and its child Finger objects will have a different ID than in an earlier frame.

Parameters

id	The ID value of a Pointable object from a previous frame.

Returns

The Pointable object with the matching ID if one exists for this hand in this frame; otherwise, an invalid Pointable object is returned.

Since

1.0

5.16.3.22 pointables()

```
LEAP_EXPORT PointableList Leap::Hand::pointables ( ) const
```

The list of Pointable objects detected in this frame that are associated with this hand, given in arbitrary order. The list will always contain 5 fingers.

Use PointableList::extended() to remove non-extended fingers from the list.

Returns

The PointableList containing all Pointable objects associated with this hand.

Since

1.0

5.16.3.23 rotationAngle() [1/2]

```
LEAP_EXPORT float Leap::Hand::rotationAngle (
            const Frame & sinceFrame ) const
```

The angle of rotation around the rotation axis derived from the change in orientation of this hand, and any associated fingers, between the current frame and the specified frame.

The returned angle is expressed in radians measured clockwise around the rotation axis (using the right-hand rule) between the start and end frames. The value is always between 0 and pi radians (0 and 180 degrees).

If a corresponding Hand object is not found in sinceFrame, or if either this frame or sinceFrame are invalid Frame objects, then the angle of rotation is zero.

Parameters

sinceFrame	The starting frame for computing the relative rotation.

Returns

A positive value representing the heuristically determined rotational change of the hand between the current frame and that specified in the sinceFrame parameter.

Since

1.0

5.16.3.24 rotationAngle() [2/2]

```
LEAP_EXPORT float Leap::Hand::rotationAngle (
            const Frame & sinceFrame,
            const Vector & axis ) const
```

The angle of rotation around the specified axis derived from the change in orientation of this hand, and any associated fingers, between the current frame and the specified frame.

The returned angle is expressed in radians measured clockwise around the rotation axis (using the right-hand rule) between the start and end frames. The value is always between -pi and pi radians (-180 and 180 degrees).

If a corresponding Hand object is not found in sinceFrame, or if either this frame or sinceFrame are invalid Frame objects, then the angle of rotation is zero.

Parameters

sinceFrame	The starting frame for computing the relative rotation.
axis	The axis to measure rotation around.

Returns

A value representing the heuristically determined rotational change of the hand between the current frame and that specified in the sinceFrame parameter around the specified axis.

Since

1.0

5.16.3.25 rotationAxis()

```
LEAP_EXPORT Vector Leap::Hand::rotationAxis (
            const Frame & sinceFrame ) const
```

The axis of rotation derived from the change in orientation of this hand, and any associated fingers, between the current frame and the specified frame.

The returned direction vector is normalized.

If a corresponding Hand object is not found in sinceFrame, or if either this frame or sinceFrame are invalid Frame objects, then this method returns a zero vector.

Parameters

sinceFrame	The starting frame for computing the relative rotation.

Returns

A normalized direction Vector representing the heuristically determined axis of rotational change of the hand between the current frame and that specified in the sinceFrame parameter.

Since

1.0

5.16.3.26 rotationMatrix()

```
LEAP_EXPORT Matrix Leap::Hand::rotationMatrix (
            const Frame & sinceFrame ) const
```

The transform matrix expressing the rotation derived from the change in orientation of this hand, and any associated fingers, between the current frame and the specified frame.

If a corresponding Hand object is not found in sinceFrame, or if either this frame or sinceFrame are invalid Frame objects, then this method returns an identity matrix.

Parameters

sinceFrame	The starting frame for computing the relative rotation.

Returns

A transformation Matrix representing the heuristically determined rotational change of the hand between the current frame and that specified in the sinceFrame parameter.

Since

1.0

5.16.3.27 rotationProbability()

```
LEAP_EXPORT float Leap::Hand::rotationProbability (
            const Frame & sinceFrame ) const
```

The estimated probability that the hand motion between the current frame and the specified frame is intended to be a rotating motion.

If a corresponding Hand object is not found in sinceFrame, or if either this frame or sinceFrame are invalid Frame objects, then this method returns zero.

Parameters

sinceFrame	The starting frame for computing the relative rotation.

Returns

A value between 0 and 1 representing the estimated probability that the hand motion between the current frame and the specified frame is intended to be a rotating motion.

Since

1.0

5.16.3.28 scaleFactor()

```
LEAP_EXPORT float Leap::Hand::scaleFactor (
            const Frame & sinceFrame ) const
```

The scale factor derived from this hand's motion between the current frame and the specified frame.

The scale factor is always positive. A value of 1.0 indicates no scaling took place. Values between 0.0 and 1.0 indicate contraction and values greater than 1.0 indicate expansion.

The Leap Motion software derives scaling from the relative inward or outward motion of a hand and its associated fingers (independent of translation and rotation).

If a corresponding Hand object is not found in sinceFrame, or if either this frame or sinceFrame are invalid Frame objects, then this method returns 1.0.

Parameters

sinceFrame	The starting frame for computing the relative scaling.

Returns

A positive value representing the heuristically determined scaling change ratio of the hand between the current frame and that specified in the sinceFrame parameter.

Since

1.0

5.16.3.29 scaleProbability()

```
LEAP_EXPORT float Leap::Hand::scaleProbability (
            const Frame & sinceFrame ) const
```

The estimated probability that the hand motion between the current frame and the specified frame is intended to be a scaling motion.

If a corresponding Hand object is not found in sinceFrame, or if either this frame or sinceFrame are invalid Frame objects, then this method returns zero.

Parameters

sinceFrame	The starting frame for computing the relative scaling.

Returns

A value between 0 and 1 representing the estimated probability that the hand motion between the current frame and the specified frame is intended to be a scaling motion.

Since

1.0

5.16.3.30 sphereCenter()

```
LEAP_EXPORT Vector Leap::Hand::sphereCenter ( ) const
```

The center of a sphere fit to the curvature of this hand.

This sphere is placed roughly as if the hand were holding a ball.

Returns

The Vector representing the center position of the sphere.

Since

1.0

5.16.3.31 sphereRadius()

```
LEAP_EXPORT float Leap::Hand::sphereRadius ( ) const
```

The radius of a sphere fit to the curvature of this hand.

This sphere is placed roughly as if the hand were holding a ball. Thus the size of the sphere decreases as the fingers are curled into a fist.

Returns

The radius of the sphere in millimeters.

Since

1.0

5.16.3.32 stabilizedPalmPosition()

```
LEAP_EXPORT Vector Leap::Hand::stabilizedPalmPosition ( ) const
```

The stabilized palm position of this Hand.

Smoothing and stabilization is performed in order to make this value more suitable for interaction with 2D content. The stabilized position lags behind the palm position by a variable amount, depending primarily on the speed of movement.

Returns

A modified palm position of this Hand object with some additional smoothing and stabilization applied.

Since

1.0

5.16.3.33 timeVisible()

```
LEAP_EXPORT float Leap::Hand::timeVisible ( ) const
```

The duration of time this Hand has been visible to the Leap Motion Controller.

Returns

The duration (in seconds) that this Hand has been tracked.

Since

1.0

5.16.3.34 tool()

```
LEAP_EXPORT Tool Leap::Hand::tool (
             int32_t id ) const
```

Tools are not associated with hands in version 2+. This function always returns an invalid Tool object.

Deprecated 2.0

5.16.3.35 tools()

```
LEAP_EXPORT ToolList Leap::Hand::tools ( ) const
```

Tools are not associated with hands in version 2+. This list is always empty.

Deprecated 2.0

5.16.3.36 toString()

```
std::string Leap::Hand::toString ( ) const  [inline]
```

A string containing a brief, human readable description of the Hand object.

Returns

A description of the Hand as a string.

Since

1.0

5.16.3.37 translation()

```
LEAP_EXPORT Vector Leap::Hand::translation (
             const Frame & sinceFrame ) const
```

The change of position of this hand between the current frame and the specified frame.

The returned translation vector provides the magnitude and direction of the movement in millimeters.

If a corresponding Hand object is not found in sinceFrame, or if either this frame or sinceFrame are invalid Frame objects, then this method returns a zero vector.

Parameters

sinceFrame	The starting frame for computing the translation.

Returns

A Vector representing the heuristically determined change in hand position between the current frame and that specified in the sinceFrame parameter.

Since

1.0

5.16.3.38 translationProbability()

```
LEAP_EXPORT float Leap::Hand::translationProbability (
             const Frame & sinceFrame ) const
```

The estimated probability that the hand motion between the current frame and the specified frame is intended to be a translating motion.

If a corresponding Hand object is not found in sinceFrame, or if either this frame or sinceFrame are invalid Frame objects, then this method returns zero.

Parameters

sinceFrame	The starting frame for computing the translation.

Returns

A value between 0 and 1 representing the estimated probability that the hand motion between the current frame and the specified frame is intended to be a translating motion.

Since

1.0

5.16.3.39 wristPosition()

```
LEAP_EXPORT Vector Leap::Hand::wristPosition ( ) const
```

The position of the wrist of this hand.

Returns

A vector containing the coordinates of the wrist position in millimeters.

Since

2.0.3

5.16.4 Friends And Related Function Documentation

5.16.4.1 operator<<

```
LEAP_EXPORT friend std::ostream& operator<< (
            std::ostream & ,
            const Hand &  )  [friend]
```

Writes a brief, human readable description of the Hand object to an output stream.

Since

1.0

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

- sample/include/Leap.h

5.17 Leap::HandList Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::HandList:

Collaboration diagram for Leap::HandList:

Public Types

- typedef ConstListIterator< HandList, Hand > const_iterator

Public Member Functions

- **HandList** (const ListBaseImplementation< Hand > &)
- LEAP_EXPORT HandList ()
- LEAP_EXPORT int count () const
- LEAP_EXPORT bool isEmpty () const
- LEAP_EXPORT Hand operator[ ] (int index) const
- LEAP_EXPORT HandList & append (const HandList &other)
- LEAP_EXPORT Hand leftmost () const
- LEAP_EXPORT Hand rightmost () const
- LEAP_EXPORT Hand frontmost () const
- LEAP_EXPORT const_iterator begin () const
- LEAP_EXPORT const_iterator end () const

Additional Inherited Members

5.17.1 Detailed Description

The HandList class represents a list of Hand objects.

Get a HandList object by calling Frame::hands().

Since

1.0

5.17.2 Member Typedef Documentation

5.17.2.1 const_iterator

```
typedef ConstListIterator<HandList, Hand> Leap::HandList::const_iterator
```

A C++ iterator type for this HandList objects.

Since

1.0

5.17.3 Constructor & Destructor Documentation

5.17.3.1 HandList()

```
LEAP_EXPORT Leap::HandList::HandList ( )
```

Constructs an empty list of hands.

Since

1.0

5.17.4 Member Function Documentation

5.17.4.1 append()

```
LEAP_EXPORT HandList& Leap::HandList::append (
                const HandList & other )
```

Appends the members of the specified HandList to this HandList.

Parameters

other	A HandList object containing Hand objects to append to the end of this HandList.

5.17.4.2 begin()

```
LEAP_EXPORT const_iterator Leap::HandList::begin ( ) const
```

The C++ iterator set to the beginning of this HandList.

Since

1.0

5.17.4.3 count()

```
LEAP_EXPORT int Leap::HandList::count ( ) const
```

Returns the number of hands in this list.

Returns

The number of hands in this list.

Since

1.0

5.17.4.4 end()

```
LEAP_EXPORT const_iterator Leap::HandList::end ( ) const
```

The C++ iterator set to the end of this HandList.

Since

1.0

5.17.4.5 frontmost()

```
LEAP_EXPORT Hand Leap::HandList::frontmost ( ) const
```

The member of the list that is farthest to the front within the standard Leap Motion frame of reference (i.e has the smallest Z coordinate).

Returns

The frontmost hand, or invalid if list is empty.

Since

1.0

5.17.4.6 isEmpty()

```
LEAP_EXPORT bool Leap::HandList::isEmpty ( ) const
```

Reports whether the list is empty.

Returns

True, if the list has no members.

Since

1.0

5.17.4.7 leftmost()

```
LEAP_EXPORT Hand Leap::HandList::leftmost ( ) const
```

The member of the list that is farthest to the left within the standard Leap Motion frame of reference (i.e has the smallest X coordinate).

Note: to determine whether a hand is the left hand, use the Hand::isLeft() function.

Returns

The leftmost hand, or invalid if list is empty.

Since

1.0

5.17.4.8 operator[]()

```
LEAP_EXPORT Hand Leap::HandList::operator[] (
             int index ) const
```

Access a list member by its position in the list.

Parameters

index	The zero-based list position index.

Returns

The Hand object at the specified index.

Since

1.0

5.17.4.9 rightmost()

```
LEAP_EXPORT Hand Leap::HandList::rightmost ( ) const
```

The member of the list that is farthest to the right within the standard Leap Motion frame of reference (i.e has the largest X coordinate).

Note: to determine whether a hand is the right hand, use the Hand::isRight() function.

Returns

The rightmost hand, or invalid if list is empty.

Since

1.0

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

- sample/include/Leap.h

5.18 Leap::Image Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Image:

Collaboration diagram for Leap::Image:

Public Types

- enum FormatType { **INFRARED** = 0 }

Public Member Functions

- **Image** (ImageImplementation ∗)
- LEAP_EXPORT Image ()
- LEAP_EXPORT int64_t sequenceId () const
- LEAP_EXPORT int32_t id () const
- LEAP_EXPORT const unsigned char ∗ data () const
- LEAP_EXPORT const float ∗ distortion () const
- void **data** (unsigned char ∗dst) const
- void **distortion** (float ∗dst) const
- void ∗ **dataPointer** () const
- void ∗ **distortionPointer** () const
- LEAP_EXPORT int width () const
- LEAP_EXPORT int height () const
- LEAP_EXPORT int bytesPerPixel () const
- LEAP_EXPORT FormatType format () const
- LEAP_EXPORT int distortionWidth () const
- LEAP_EXPORT int distortionHeight () const
- LEAP_EXPORT float rayOffsetX () const
- LEAP_EXPORT float rayOffsetY () const
- LEAP_EXPORT float rayScaleX () const
- LEAP_EXPORT float rayScaleY () const
- LEAP_EXPORT Vector rectify (const Vector &uv) const
- LEAP_EXPORT Vector warp (const Vector &xy) const
- LEAP_EXPORT int64_t timestamp () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT bool operator== (const Image &) const
- LEAP_EXPORT bool operator!= (const Image &) const
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const Image & invalid ()

Friends

- LEAP_EXPORT friend std::ostream & operator<< (std::ostream &, const Image &)

Additional Inherited Members

5.18.1 Detailed Description

The Image class represents a single image from one of the Leap Motion cameras.

In addition to image data, the Image object provides a distortion map for correcting lens distortion.

Note that Image objects can be invalid, which means that they do not contain valid image data. Get valid Image objects from Frame::frames(). Test for validity with the Image::isValid() function.

Since

2.1.0

5.18.2 Member Enumeration Documentation

5.18.2.1 FormatType

`enum Leap::Image::FormatType`

Enumerates the possible image formats.

The Image::format() function returns an item from the FormatType enumeration.

Since

2.2.0

5.18.3 Constructor & Destructor Documentation

5.18.3.1 Image()

`LEAP_EXPORT Leap::Image::Image ( )`

Constructs a Image object.

An uninitialized image is considered invalid. Get valid Image objects from a ImageList object obtained from the Frame::images() method.

Since

2.1.0

5.18.4 Member Function Documentation

5.18.4.1 bytesPerPixel()

`LEAP_EXPORT int Leap::Image::bytesPerPixel ( ) const`

The number of bytes per pixel.

Use this value along with `Image::width()` and `Image:::height()` to calculate the size of the data buffer.

Since

2.2.0

5.18.4.2 data()

```
LEAP_EXPORT const unsigned char* Leap::Image::data ( ) const
```

The image data.

The image data is a set of 8-bit intensity values. The buffer is `Image::width()` * `Image::height()` * `Image::bytesPerPixel()` bytes long.

Returns

The array of unsigned char containing the sensor brightness values.

Since

2.1.0

5.18.4.3 distortion()

```
LEAP_EXPORT const float* Leap::Image::distortion ( ) const
```

The distortion calibration map for this image.

The calibration map is a 64x64 grid of points. Each point is defined by a pair of 32-bit floating point values. Each point in the map represents a ray projected into the camera. The value of a grid point defines the pixel in the image data containing the brightness value produced by the light entering along the corresponding ray. By interpolating between grid data points, you can find the brightness value for any projected ray. Grid values that fall outside the range [0..1] do not correspond to a value in the image data and those points should be ignored.

The calibration map can be used to render an undistorted image as well as to find the true angle from the camera to a feature in the raw image. The distortion map itself is designed to be used with GLSL shader programs. In non-realtime contexts, it may be more convenient to use the Image::rectify() and Image::warp() functions.

If using shaders is not possible, you can use the distortion map directly. This can be faster than using the `warp()` function, if carefully optimized:

Distortion is caused by the lens geometry as well as imperfections in the lens and sensor window. The calibration map is created by the calibration process run for each device at the factory (and which can be rerun by the user).

Note, in a future release, there may be two distortion maps per image; one containing the horizontal values and the other containing the vertical values.

Returns

The float array containing the camera lens distortion map.

Since

2.1.0

5.18.4.4 distortionHeight()

```
LEAP_EXPORT int Leap::Image::distortionHeight ( ) const
```

The distortion map height.

Currently fixed at 64.

Since

2.1.0

5.18.4.5 distortionWidth()

```
LEAP_EXPORT int Leap::Image::distortionWidth ( ) const
```

The stride of the distortion map.

Since each point on the 64x64 element distortion map has two values in the buffer, the stride is 2 times the size of the grid. (Stride is currently fixed at $2 * 64 = 128$).

Since

2.1.0

5.18.4.6 format()

```
LEAP_EXPORT FormatType Leap::Image::format ( ) const
```

The image format.

Since

2.2.0

5.18.4.7 height()

```
LEAP_EXPORT int Leap::Image::height ( ) const
```

The image height.

Since

2.1.0

5.18.4.8 id()

```
LEAP_EXPORT int32_t Leap::Image::id ( ) const
```

The image ID.

Images with ID of 0 are from the left camera; those with an ID of 1 are from the right camera (with the device in its standard operating position with the green LED facing the operator).

Since

2.1.0

5.18.4.9 invalid()

```
static LEAP_EXPORT const Image& Leap::Image::invalid ( )  [static]
```

Returns an invalid Image object.

You can use the instance returned by this function in comparisons testing whether a given Image instance is valid or invalid. (You can also use the Image::isValid() function.)

Returns

The invalid Image instance.

Since

2.1.0

5.18.4.10 isValid()

```
LEAP_EXPORT bool Leap::Image::isValid ( ) const
```

Reports whether this Image instance contains valid data.

Returns

true, if and only if the image is valid.

Since

2.1.0

5.18.4.11 operator"!=()

```
LEAP_EXPORT bool Leap::Image::operator!= (
             const Image & ) const
```

Compare Image object inequality.

Two Image objects are equal if and only if both Image objects represent the exact same Image and both Images are valid.

Since

2.1.0

5.18.4.12 operator==()

```
LEAP_EXPORT bool Leap::Image::operator== (
             const Image & ) const
```

Compare Image object equality.

Two Image objects are equal if and only if both Image objects represent the exact same Image and both Images are valid.

Since

2.1.0

5.18.4.13 rayOffsetX()

```
LEAP_EXPORT float Leap::Image::rayOffsetX ( ) const
```

The horizontal ray offset.

Used to convert between normalized coordinates in the range [0..1] and the ray slope range [-4..4].

Since

2.1.0

5.18.4.14 rayOffsetY()

```
LEAP_EXPORT float Leap::Image::rayOffsetY ( ) const
```

The vertical ray offset.

Used to convert between normalized coordinates in the range [0..1] and the ray slope range [-4..4].

Since

2.1.0

5.18.4.15 rayScaleX()

```
LEAP_EXPORT float Leap::Image::rayScaleX ( ) const
```

The horizontal ray scale factor.

Used to convert between normalized coordinates in the range [0..1] and the ray slope range [-4..4].

Since

2.1.0

5.18.4.16 rayScaleY()

```
LEAP_EXPORT float Leap::Image::rayScaleY ( ) const
```

The vertical ray scale factor.

Used to convert between normalized coordinates in the range [0..1] and the ray slope range [-4..4].

Since

2.1.0

5.18.4.17 rectify()

```
LEAP_EXPORT Vector Leap::Image::rectify (
             const Vector & uv ) const
```

Provides the corrected camera ray intercepting the specified point on the image.

Given a point on the image, `rectify()` corrects for camera distortion and returns the true direction from the camera to the source of that image point within the Leap Motion field of view.

This direction vector has an x and y component [x, y, 0], with the third element always zero. Note that this vector uses the 2D camera coordinate system where the x-axis parallels the longer (typically horizontal) dimension and the y-axis parallels the shorter (vertical) dimension. The camera coordinate system does not correlate to the 3D Leap Motion coordinate system.

Parameters

uv	A Vector containing the position of a pixel in the image.

Returns

A Vector containing the ray direction (the z-component of the vector is always 0).

Since

2.1.0

5.18.4.18 sequenceId()

```
LEAP_EXPORT int64_t Leap::Image::sequenceId ( ) const
```

The image sequence ID.

Since

2.2.1

5.18.4.19 timestamp()

```
LEAP_EXPORT int64_t Leap::Image::timestamp ( ) const
```

Returns a timestamp indicating when this frame began being captured on the device.

Since

2.2.7

5.18.4.20 toString()

```
std::string Leap::Image::toString ( ) const  [inline]
```

A string containing a brief, human readable description of the Image object.

Returns

A description of the Image as a string.

Since

2.1.0

5.18.4.21 warp()

```
LEAP_EXPORT Vector Leap::Image::warp (
             const Vector & xy ) const
```

Provides the point in the image corresponding to a ray projecting from the camera.

Given a ray projected from the camera in the specified direction, `warp()` corrects for camera distortion and returns the corresponding pixel coordinates in the image.

The ray direction is specified in relationship to the camera. The first vector element corresponds to the "horizontal" view angle; the second corresponds to the "vertical" view angle.

The `warp()` function returns pixel coordinates outside of the image bounds if you project a ray toward a point for which there is no recorded data.

`warp()` is typically not fast enough for realtime distortion correction. For better performance, use a shader program exectued on a GPU.

Parameters

xy	A Vector containing the ray direction.

Returns

A Vector containing the pixel coordinates [x, y, 0] (with z always zero).

Since

2.1.0

5.18.4.22 width()

```
LEAP_EXPORT int Leap::Image::width ( ) const
```

The image width.

Since

2.1.0

5.18.5 Friends And Related Function Documentation

5.18.5.1 operator<<

```
LEAP_EXPORT friend std::ostream& operator<< (
             std::ostream & ,
             const Image & )  [friend]
```

Writes a brief, human readable description of the Image object.

Since

2.1.0

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

- sample/include/Leap.h

5.19 Leap::ImageList Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::ImageList:

Collaboration diagram for Leap::ImageList:

Public Types

- typedef ConstListIterator< ImageList, Image > const_iterator

Public Member Functions

- **ImageList** (const ListBaseImplementation< Image > &)
- LEAP_EXPORT ImageList ()
- LEAP_EXPORT int count () const
- LEAP_EXPORT bool isEmpty () const
- LEAP_EXPORT Image operator[ ] (int index) const
- LEAP_EXPORT ImageList & append (const ImageList &other)
- LEAP_EXPORT const_iterator begin () const
- LEAP_EXPORT const_iterator end () const

Additional Inherited Members

5.19.1 Detailed Description

The ImageList class represents a list of Image objects.

Get the ImageList object associated with the a Frame of tracking data by calling Frame::images(). Get the most recent set of images, which can be newer than the images used to create the current frame, by calling Controller↩ ::images().

Since

2.1.0

5.19.2 Member Typedef Documentation

5.19.2.1 const_iterator

```
typedef ConstListIterator<ImageList, Image> Leap::ImageList::const_iterator
```

A C++ iterator type for this ImageList objects.

Since

2.1.0

5.19.3 Constructor & Destructor Documentation

5.19.3.1 ImageList()

```
LEAP_EXPORT Leap::ImageList::ImageList ( )
```

Constructs an empty list of images.

Since

2.1.0

5.19.4 Member Function Documentation

5.19.4.1 append()

```
LEAP_EXPORT ImageList& Leap::ImageList::append (
             const ImageList & other )
```

Appends the members of the specified ImageList to this ImageList.

Parameters

other	A ImageList object containing Image objects to append to the end of this ImageList.

Since

2.1.0

5.19.4.2 begin()

```
LEAP_EXPORT const_iterator Leap::ImageList::begin ( ) const
```

The C++ iterator set to the beginning of this ImageList.

Since

2.1.0

5.19.4.3 count()

```
LEAP_EXPORT int Leap::ImageList::count ( ) const
```

The number of images in this list.

Returns

The number of images in this list.

Since

2.1.0

5.19.4.4 end()

```
LEAP_EXPORT const_iterator Leap::ImageList::end ( ) const
```

The C++ iterator set to the end of this ImageList.

Since

2.1.0

5.19.4.5 isEmpty()

```
LEAP_EXPORT bool Leap::ImageList::isEmpty ( ) const
```

Reports whether the list is empty.

Returns

True, if the list has no members.

Since

2.1.0

5.19.4.6 operator[]()

```
LEAP_EXPORT Image Leap::ImageList::operator[ ] (
             int index ) const
```

Access a list member by its position in the list.

Parameters

index	The zero-based list position index.

Returns

The Image object at the specified index.

Since

2.1.0

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

- sample/include/Leap.h

5.20 Leap::Interface::Implementation Struct Reference

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

- sample/include/Leap.h

5.21 Leap::InteractionBox Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::InteractionBox:

Collaboration diagram for Leap::InteractionBox:

Public Member Functions

- **InteractionBox** (InteractionBoxImplementation ∗)
- LEAP_EXPORT Vector normalizePoint (const Vector &position, bool clamp=true) const
- LEAP_EXPORT Vector denormalizePoint (const Vector &normalizedPosition) const
- LEAP_EXPORT Vector center () const
- LEAP_EXPORT float width () const
- LEAP_EXPORT float height () const
- LEAP_EXPORT float depth () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT bool operator== (const InteractionBox &) const
- LEAP_EXPORT bool operator!= (const InteractionBox &) const
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const InteractionBox & invalid ()

Friends

- LEAP_EXPORT friend std::ostream & operator<< (std::ostream &, const InteractionBox &)

Additional Inherited Members

5.21.1 Detailed Description

The InteractionBox class represents a box-shaped region completely within the field of view of the Leap Motion controller.

The interaction box is an axis-aligned rectangular prism and provides normalized coordinates for hands, fingers, and tools within this box. The InteractionBox class can make it easier to map positions in the Leap Motion coordinate system to 2D or 3D coordinate systems used for application drawing.

The InteractionBox region is defined by a center and dimensions along the x, y, and z axes.

Get an InteractionBox object from a Frame object.

Since

1.0

5.21.2 Member Function Documentation

5.21.2.1 center()

```
LEAP_EXPORT Vector Leap::InteractionBox::center ( ) const
```

The center of the InteractionBox in device coordinates (millimeters). This point is equidistant from all sides of the box.

Returns

The InteractionBox center in device coordinates.

Since

1.0

5.21.2.2 denormalizePoint()

```
LEAP_EXPORT Vector Leap::InteractionBox::denormalizePoint (
              const Vector & normalizedPosition ) const
```

Converts a position defined by normalized InteractionBox coordinates into device coordinates in millimeters.

This function performs the inverse of normalizePoint().

Parameters

normalizedPosition	The input position in InteractionBox coordinates.

Returns

The corresponding denormalized position in device coordinates.

Since

1.0

5.21.2.3 depth()

```
LEAP_EXPORT float Leap::InteractionBox::depth ( ) const
```

The depth of the InteractionBox in millimeters, measured along the z-axis.

Returns

The InteractionBox depth in millimeters.

Since

1.0

5.21.2.4 height()

```
LEAP_EXPORT float Leap::InteractionBox::height ( ) const
```

The height of the InteractionBox in millimeters, measured along the y-axis.

Returns

The InteractionBox height in millimeters.

Since

1.0

5.21.2.5 invalid()

```
static LEAP_EXPORT const InteractionBox& Leap::InteractionBox::invalid ( )  [static]
```

Returns an invalid InteractionBox object.

You can use the instance returned by this function in comparisons testing whether a given InteractionBox instance is valid or invalid. (You can also use the InteractionBox::isValid() function.)

Returns

The invalid InteractionBox instance.

Since

1.0

5.21.2.6 isValid()

```
LEAP_EXPORT bool Leap::InteractionBox::isValid ( ) const
```

Reports whether this is a valid InteractionBox object.

Returns

True, if this InteractionBox object contains valid data.

Since

1.0

5.21.2.7 normalizePoint()

```
LEAP_EXPORT Vector Leap::InteractionBox::normalizePoint (
             const Vector & position,
             bool clamp = true ) const
```

Normalizes the coordinates of a point using the interaction box.

Coordinates from the Leap Motion frame of reference (millimeters) are converted to a range of [0..1] such that the minimum value of the InteractionBox maps to 0 and the maximum value of the InteractionBox maps to 1.

Parameters

position	The input position in device coordinates.
clamp	Whether or not to limit the output value to the range [0,1] when the input position is outside the InteractionBox. Defaults to true.

Returns

The normalized position.

Since

1.0

5.21.2.8 operator"!=()

```
LEAP_EXPORT bool Leap::InteractionBox::operator!= (
             const InteractionBox &  ) const
```

Compare InteractionBox object inequality.

Two InteractionBox objects are equal if and only if both InteractionBox objects represent the exact same InteractionBox and both InteractionBoxes are valid.

Since

1.0

5.21.2.9 operator==()

```
LEAP_EXPORT bool Leap::InteractionBox::operator== (
             const InteractionBox &  ) const
```

Compare InteractionBox object equality.

Two InteractionBox objects are equal if and only if both InteractionBox objects represent the exact same InteractionBox and both InteractionBoxes are valid.

Since

1.0

5.21.2.10 toString()

```
std::string Leap::InteractionBox::toString ( ) const  [inline]
```

A string containing a brief, human readable description of the InteractionBox object.

Returns

A description of the InteractionBox as a string.

Since

1.0

5.21.2.11 width()

```
LEAP_EXPORT float Leap::InteractionBox::width ( ) const
```

The width of the InteractionBox in millimeters, measured along the x-axis.

Returns

The InteractionBox width in millimeters.

Since

1.0

5.21.3 Friends And Related Function Documentation

5.21.3.1 operator<<

```
LEAP_EXPORT friend std::ostream& operator<< (
             std::ostream & ,
             const InteractionBox &  )  [friend]
```

Writes a brief, human readable description of the InteractionBox object.

Since

1.0

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

- sample/include/Leap.h

5.22 Leap::Interface Class Reference

Inheritance diagram for Leap::Interface:

Classes

- struct Implementation

Protected Member Functions

- LEAP_EXPORT **Interface** (void ∗owner)
- LEAP_EXPORT **Interface** (Implementation ∗reference, void ∗owner)
- LEAP_EXPORT **Interface** (const Interface &rhs)
- **Interface** (class SharedObject ∗object)
- LEAP_EXPORT Interface & **operator=** (const Interface &rhs)
- template<typename T >
  T ∗ **get** () const

Static Protected Member Functions

- static LEAP_EXPORT void **deleteCString** (const char ∗cstr)

Protected Attributes

- class SharedObject ∗ **m_object**

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

- sample/include/Leap.h

5.23 Leap::KeyTapGesture Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::KeyTapGesture:

Collaboration diagram for Leap::KeyTapGesture:

Public Member Functions

- LEAP_EXPORT KeyTapGesture ()
- LEAP_EXPORT KeyTapGesture (const Gesture &rhs)
- LEAP_EXPORT Vector position () const
- LEAP_EXPORT Vector direction () const
- LEAP_EXPORT float progress () const
- LEAP_EXPORT Pointable pointable () const

Static Public Member Functions

- static Type classType ()

Additional Inherited Members

5.23.1 Detailed Description

The KeyTapGesture class represents a tapping gesture by a finger or tool.

A key tap gesture is recognized when the tip of a finger rotates down toward the palm and then springs back to approximately the original position, as if tapping. The tapping finger must pause briefly before beginning the tap.

Important: To use key tap gestures in your application, you must enable recognition of the key tap gesture. You can enable recognition with:

Key tap gestures are discrete. The KeyTapGesture object representing a tap always has the state, STATE_STOP. Only one KeyTapGesture object is created for each key tap gesture recognized.

You can set the minimum finger movement and velocity required for a movement to be recognized as a key tap as well as adjust the detection window for evaluating the movement using the config attribute of a connected Controller object. Use the following configuration keys to configure key tap recognition:

==================================== ========== ============= ======= Key string Value type Default value Units ==================================== ========== ============= ======= Gesture.KeyTap.MinDownVelocity float 50 mm/s Gesture.KeyTap.HistorySeconds float 0.1 s Gesture.KeyTap.Min←Distance float 3.0 mm ==================================== ========== ============= =======

The following example demonstrates how to set the key tap configuration parameters:

The Controller object must be connected to the Leap Motion service/daemon before setting the configuration parameters.

Since

1.0

5.23.2 Constructor & Destructor Documentation

5.23.2.1 KeyTapGesture() [1/2]

```
LEAP_EXPORT Leap::KeyTapGesture::KeyTapGesture ( )
```

Constructs a new KeyTapGesture object.

An uninitialized KeyTapGesture object is considered invalid. Get valid instances of the KeyTapGesture class from a Frame object.

Since

1.0

5.23.2.2 KeyTapGesture() [2/2]

```
LEAP_EXPORT Leap::KeyTapGesture::KeyTapGesture (
             const Gesture & rhs )
```

Constructs a KeyTapGesture object from an instance of the Gesture class.

Parameters

rhs	The Gesture instance to specialize. This Gesture instance must be a KeyTapGesture object.

Since

1.0

5.23.3 Member Function Documentation

5.23.3.1 classType()

```
static Type Leap::KeyTapGesture::classType ( )  [inline], [static]
```

The key tap gesture type.

Returns

Type The type value designating a key tap gesture.

Since

1.0

5.23.3.2 direction()

```
LEAP_EXPORT Vector Leap::KeyTapGesture::direction ( ) const
```

The direction of finger tip motion.

Returns

Vector A unit direction vector if the finger tip is moving; otherwise, a zero-vector.

Since

1.0

5.23.3.3 pointable()

```
LEAP_EXPORT Pointable Leap::KeyTapGesture::pointable ( ) const
```

The finger performing the key tap gesture.

Returns

Pointable A Pointable object representing the tapping finger.

Since

1.0

5.23.3.4 position()

```
LEAP_EXPORT Vector Leap::KeyTapGesture::position ( ) const
```

The position where the key tap is registered.

Returns

Vector A Vector containing the coordinates of tap location.

Since

1.0

5.23.3.5 progress()

```
LEAP_EXPORT float Leap::KeyTapGesture::progress ( ) const
```

The progress value is always 1.0 for a key tap gesture.

Returns

float The value 1.0.

Since

1.0

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

- sample/include/Leap.h

5.24 Leap::ListBaseImplementation$<$ T $>$ Class Template Reference

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

- sample/include/Leap.h

5.25 Leap::Listener Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Listener:

Public Member Functions

- LEAP_EXPORT Listener ()
- virtual LEAP_EXPORT ∼Listener ()
- virtual LEAP_EXPORT void onInit (const Controller &)
- virtual LEAP_EXPORT void onConnect (const Controller &)
- virtual LEAP_EXPORT void onDisconnect (const Controller &)
- virtual LEAP_EXPORT void onExit (const Controller &)
- virtual LEAP_EXPORT void onFrame (const Controller &)
- virtual LEAP_EXPORT void onFocusGained (const Controller &)
- virtual LEAP_EXPORT void onFocusLost (const Controller &)
- virtual LEAP_EXPORT void onServiceConnect (const Controller &)
- virtual LEAP_EXPORT void onServiceDisconnect (const Controller &)
- virtual LEAP_EXPORT void onDeviceChange (const Controller &)
- virtual LEAP_EXPORT void onImages (const Controller &)

5.25.1 Detailed Description

The Listener class defines a set of callback functions that you can override in a subclass to respond to events dispatched by the Controller object.

To handle Leap Motion events, create an instance of a Listener subclass and assign it to the Controller instance. The Controller calls the relevant Listener callback function when an event occurs, passing in a reference to itself. You do not have to implement callbacks for events you do not want to handle.

The Controller object calls these Listener functions from a thread created by the Leap Motion library, not the thread used to create or set the Listener instance.

Since

1.0

5.25.2 Constructor & Destructor Documentation

5.25.2.1 Listener()

```
LEAP_EXPORT Leap::Listener::Listener ( )  [inline]
```

Constructs a Listener object.

Since

1.0

5.25.2.2 ∼Listener()

```
virtual LEAP_EXPORT Leap::Listener::∼Listener ( )  [inline], [virtual]
```

Destroys this Listener object.

5.25.3 Member Function Documentation

5.25.3.1 onConnect()

```
virtual LEAP_EXPORT void Leap::Listener::onConnect (
            const Controller &  )  [inline], [virtual]
```

Called when the Controller object connects to the Leap Motion software and the Leap Motion hardware device is plugged in, or when this Listener object is added to a Controller that is already connected.

When this callback is invoked, Controller::isServiceConnected is true, Controller::devices() is not empty, and, for at least one of the Device objects in the list, Device::isStreaming() is true.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented in SampleListener, and SampleListener.

5.25.3.2 onDeviceChange()

```
virtual LEAP_EXPORT void Leap::Listener::onDeviceChange (
            const Controller & )  [inline], [virtual]
```

Called when a Leap Motion controller plugged in, unplugged, or the device changes state.

State changes include changes in frame rate and entering or leaving "robust" mode. Note that there is currently no way to query whether a device is in robust mode. You can use Frame::currentFramerate() to get the framerate.

Parameters

controller	The Controller object invoking this callback function.

Since

1.2

Reimplemented in SampleListener, and SampleListener.

5.25.3.3 onDisconnect()

```
virtual LEAP_EXPORT void Leap::Listener::onDisconnect (
            const Controller & )  [inline], [virtual]
```

Called when the Controller object disconnects from the Leap Motion software or the Leap Motion hardware is unplugged. The controller can disconnect when the Leap Motion device is unplugged, the user shuts the Leap Motion software down, or the Leap Motion software encounters an unrecoverable error.

Note: When you launch a Leap-enabled application in a debugger, the Leap Motion library does not disconnect from the application. This is to allow you to step through code without losing the connection because of time outs.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented in SampleListener, and SampleListener.

5.25.3.4 onExit()

```
virtual LEAP_EXPORT void Leap::Listener::onExit (
            const Controller &  )  [inline], [virtual]
```

Called when this Listener object is removed from the Controller or the Controller instance is destroyed.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented in SampleListener, and SampleListener.

5.25.3.5 onFocusGained()

```
virtual LEAP_EXPORT void Leap::Listener::onFocusGained (
            const Controller &  )  [inline], [virtual]
```

Called when this application becomes the foreground application.

Only the foreground application receives tracking data from the Leap Motion Controller. This function is only called when the controller object is in a connected state.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented in SampleListener, and SampleListener.

5.25.3.6 onFocusLost()

```
virtual LEAP_EXPORT void Leap::Listener::onFocusLost (
             const Controller &  )  [inline], [virtual]
```

Called when this application loses the foreground focus.

Only the foreground application receives tracking data from the Leap Motion Controller. This function is only called when the controller object is in a connected state.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented in SampleListener, and SampleListener.

5.25.3.7 onFrame()

```
virtual LEAP_EXPORT void Leap::Listener::onFrame (
             const Controller &  )  [inline], [virtual]
```

Called when a new frame of hand and finger tracking data is available. Access the new frame data using the Controller::frame() function.

Note, the Controller skips any pending onFrame events while your onFrame handler executes. If your implementation takes too long to return, one or more frames can be skipped. The Controller still inserts the skipped frames into the frame history. You can access recent frames by setting the history parameter when calling the Controller↩ ::frame() function. You can determine if any pending onFrame events were skipped by comparing the ID of the most recent frame with the ID of the last received frame.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented in SampleListener, and SampleListener.

5.25.3.8 onImages()

```
virtual LEAP_EXPORT void Leap::Listener::onImages (
            const Controller &  )  [inline], [virtual]
```

Called when new images are available. Access the new frame data using the Controller::images() function.

Parameters

controller	The Controller object invoking this callback function.

Since

2.2.1

5.25.3.9 onInit()

```
virtual LEAP_EXPORT void Leap::Listener::onInit (
            const Controller &  )  [inline], [virtual]
```

Called once, when this Listener object is newly added to a Controller.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented in SampleListener, and SampleListener.

5.25.3.10 onServiceConnect()

```
virtual LEAP_EXPORT void Leap::Listener::onServiceConnect (
            const Controller & ) [inline], [virtual]
```

Called when the Leap Motion daemon/service connects to your application Controller.

Parameters

controller	The Controller object invoking this callback function.

Since

1.2

Reimplemented in SampleListener, and SampleListener.

5.25.3.11 onServiceDisconnect()

```
virtual LEAP_EXPORT void Leap::Listener::onServiceDisconnect (
            const Controller & ) [inline], [virtual]
```

Called if the Leap Motion daemon/service disconnects from your application Controller.

Normally, this callback is not invoked. It is only called if some external event or problem shuts down the service or otherwise interrupts the connection.

Parameters

controller	The Controller object invoking this callback function.

Since

1.2

Reimplemented in SampleListener, and SampleListener.

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

- sample/include/Leap.h

5.26 Leap::Mask Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Mask:

Collaboration diagram for Leap::Mask:

Public Member Functions

- **Mask** (MaskImplementation ∗)
- LEAP_EXPORT Mask ()
- LEAP_EXPORT int64_t sequenceId () const
- LEAP_EXPORT int32_t id () const
- LEAP_EXPORT const unsigned char ∗ data () const
- void **data** (unsigned char ∗dst) const
- void ∗ **dataPointer** () const
- LEAP_EXPORT int width () const
- LEAP_EXPORT int height () const
- LEAP_EXPORT int offsetX () const
- LEAP_EXPORT int offsetY () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT bool operator== (const Mask &) const
- LEAP_EXPORT bool operator!= (const Mask &) const
- std::string **toString** () const

Static Public Member Functions

- static LEAP_EXPORT const Mask & invalid ()

Friends

- LEAP_EXPORT friend std::ostream & operator<< (std::ostream &, const Mask &)

Additional Inherited Members

5.26.1 Detailed Description

Note: This class is an experimental API for internal use only. It may be removed without warning.

A bitmap mask defining areas of an image in which a finger or part of a hand is in front of the tracked quad. The mask is a subset of the camera image containing a the region including the quad. Pixels in the mask representing the hand have the value 255. Pixels in the rest of the mask have the value 0.

Two masks are provided for every Leap Motion frame. The mask with the id of 0 is for the left image. The right image has id 1.

The mask corresponds to the uncorrected image from the camera sensor. If you correct the image for distortion before displaying it, you should also correct the mask.

Since

2.2.6

5.26.2 Constructor & Destructor Documentation

5.26.2.1 Mask()

```
LEAP_EXPORT Leap::Mask::Mask ( )
```

Constructs a new Mask object. Do not use. Get Mask objects from TrackedQuad.

Since

2.2.6

5.26.3 Member Function Documentation

5.26.3.1 data()

```
LEAP_EXPORT const unsigned char* Leap::Mask::data ( ) const
```

The pixels of the mask.

Pixels with the value of 255 represent areas of the image where a finger or part of a hand is in front of the quad. The rest of the mask has the value 0.

Since

2.2.6

5.26.3.2 height()

```
LEAP_EXPORT int Leap::Mask::height ( ) const
```

The height of the mask in Image pixels.

Since

2.2.6

5.26.3.3 id()

```
LEAP_EXPORT int32_t Leap::Mask::id ( ) const
```

An id indicating whether the mask goes with the left (0) or right (1) image.

Since

2.2.6

5.26.3.4 invalid()

```
static LEAP_EXPORT const Mask& Leap::Mask::invalid ( ) [static]
```

An invalid Mask object.

Since

2.2.6

5.26.3.5 isValid()

```
LEAP_EXPORT bool Leap::Mask::isValid ( ) const
```

Reports whether this is a valid Mask object.

Since

2.2.6

5.26.3.6 offsetX()

```
LEAP_EXPORT int Leap::Mask::offsetX ( ) const
```

The offset of the mask from the left edge of the Image in pixels.

Since

2.2.6

5.26.3.7 offsetY()

```
LEAP_EXPORT int Leap::Mask::offsetY ( ) const
```

The offset of the mask from the top edge of the Image in pixels.

Since

2.2.6

5.26.3.8 operator"!=()

```
LEAP_EXPORT bool Leap::Mask::operator!= (
             const Mask &  ) const
```

Compares two Mask objects for inequality.

Since

2.2.6

5.26.3.9 operator==()

```
LEAP_EXPORT bool Leap::Mask::operator== (
             const Mask &  ) const
```

Compares two Mask objects for equality.

Since

2.2.6

5.26.3.10 sequenceId()

```
LEAP_EXPORT int64_t Leap::Mask::sequenceId ( ) const
```

An id value based on the sequence in which the mask is produced. Corresponds to the Image sequence id.

Since

2.2.6

5.26.3.11 width()

```
LEAP_EXPORT int Leap::Mask::width ( ) const
```

The width of the mask in Image pixels.

Since

2.2.6

5.26.4 Friends And Related Function Documentation

5.26.4.1 operator<<

```
LEAP_EXPORT friend std::ostream& operator<< (
            std::ostream & ,
            const Mask &  )  [friend]
```

Writes a brief, human readable description of the Mask object.

Since

2.2.6

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

- sample/include/Leap.h

5.27 Leap::MaskList Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::MaskList:

Collaboration diagram for Leap::MaskList:

Public Types

- typedef ConstListIterator< MaskList, Mask > **const_iterator**

Public Member Functions

- **MaskList** (const ListBaseImplementation< Mask > &)
- LEAP_EXPORT MaskList ()
- LEAP_EXPORT int count () const
- LEAP_EXPORT bool isEmpty () const
- LEAP_EXPORT Mask operator[ ] (int index) const
- LEAP_EXPORT MaskList & append (const MaskList &other)
- LEAP_EXPORT const_iterator begin () const
- LEAP_EXPORT const_iterator end () const

Additional Inherited Members

5.27.1 Detailed Description

Note: This class is an experimental API for internal use only. It may be removed without warning.

A list containing Mask objects.

Since

2.2.6

5.27.2 Constructor & Destructor Documentation

5.27.2.1 MaskList()

```
LEAP_EXPORT Leap::MaskList::MaskList ( )
```

Constructs an empty list for Mask objects.

Since

2.2.6

5.27.3 Member Function Documentation

5.27.3.1 append()

```
LEAP_EXPORT MaskList& Leap::MaskList::append (
             const MaskList & other )
```

Appends the contents of another list of masks to this one.

Since

2.2.6

5.27.3.2 begin()

```
LEAP_EXPORT const_iterator Leap::MaskList::begin ( ) const
```

A list iterator set to the beginning of the list.

Since

2.2.6

5.27.3.3 count()

```
LEAP_EXPORT int Leap::MaskList::count ( ) const
```

The number of masks in this list.

Since

2.2.6

5.27.3.4 end()

```
LEAP_EXPORT const_iterator Leap::MaskList::end ( ) const
```

A list iterator set to the end of the list.

Since

2.2.6

5.27.3.5 isEmpty()

```
LEAP_EXPORT bool Leap::MaskList::isEmpty ( ) const
```

Reports whether this list is empty.

Since

2.2.6

5.27.3.6 operator[]()

```
LEAP_EXPORT Mask Leap::MaskList::operator[] (
            int index ) const
```

The MaskList supports array indexing.

Since

2.2.6

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

- sample/include/Leap.h

5.28 Leap::Matrix Struct Reference

```
#include <LeapMath.h>
```

Collaboration diagram for Leap::Matrix:

Public Member Functions

- Matrix ()
- Matrix (const Matrix &other)
- Matrix (const Vector &_xBasis, const Vector &_yBasis, const Vector &_zBasis)
- Matrix (const Vector &_xBasis, const Vector &_yBasis, const Vector &_zBasis, const Vector &_origin)
- Matrix (const Vector &axis, float angleRadians)
- Matrix (const Vector &axis, float angleRadians, const Vector &translation)
- void setRotation (const Vector &axis, float angleRadians)
- Vector transformPoint (const Vector &in) const
- Vector transformDirection (const Vector &in) const
- Matrix rigidInverse () const
- Matrix operator∗ (const Matrix &other) const
- Matrix & operator∗= (const Matrix &other)
- bool operator== (const Matrix &other) const
- bool operator!= (const Matrix &other) const
- template<typename Matrix3x3Type >
  const Matrix3x3Type toMatrix3x3 () const
- template<typename Matrix4x4Type >
  const Matrix4x4Type toMatrix4x4 () const
- template<typename T >
  T ∗ toArray3x3 (T ∗output) const
- FloatArray toArray3x3 () const
- template<typename T >
  T ∗ toArray4x4 (T ∗output) const
- FloatArray toArray4x4 () const
- std::string toString () const

Static Public Member Functions

- static const Matrix & identity ()

Public Attributes

- Vector xBasis
- Vector yBasis
- Vector zBasis
- Vector origin

Friends

- std::ostream & operator<< (std::ostream &out, const Matrix &matrix)

5.28.1 Detailed Description

The Matrix struct represents a transformation matrix.

To use this struct to transform a Vector, construct a matrix containing the desired transformation and then use the Matrix::transformPoint() or Matrix::transformDirection() functions to apply the transform.

Transforms can be combined by multiplying two or more transform matrices using the $*$ operator.

Since

1.0

5.28.2 Constructor & Destructor Documentation

5.28.2.1 Matrix() [1/6]

```
Leap::Matrix::Matrix ( )  [inline]
```

Constructs an identity transformation matrix.

Since

1.0

5.28.2.2 Matrix() [2/6]

```
Leap::Matrix::Matrix (
            const Matrix & other )  [inline]
```

Constructs a copy of the specified Matrix object.

Since

1.0

5.28.2.3 Matrix() [3/6]

```
Leap::Matrix::Matrix (
            const Vector & _xBasis,
            const Vector & _yBasis,
            const Vector & _zBasis )  [inline]
```

Constructs a transformation matrix from the specified basis vectors.

Parameters

_xBasis	A Vector specifying rotation and scale factors for the x-axis.
_yBasis	A Vector specifying rotation and scale factors for the y-axis.
_zBasis	A Vector specifying rotation and scale factors for the z-axis.

Since

1.0

5.28.2.4 Matrix() [4/6]

```
Leap::Matrix::Matrix (
            const Vector & _xBasis,
            const Vector & _yBasis,
            const Vector & _zBasis,
            const Vector & _origin )  [inline]
```

Constructs a transformation matrix from the specified basis and translation vectors.

Parameters

_xBasis	A Vector specifying rotation and scale factors for the x-axis.
_yBasis	A Vector specifying rotation and scale factors for the y-axis.
_zBasis	A Vector specifying rotation and scale factors for the z-axis.
_origin	A Vector specifying translation factors on all three axes.

Since

1.0

5.28.2.5 Matrix() [5/6]

```
Leap::Matrix::Matrix (
            const Vector & axis,
            float angleRadians )  [inline]
```

Constructs a transformation matrix specifying a rotation around the specified vector.

Parameters

axis	A Vector specifying the axis of rotation.
angleRadians	The amount of rotation in radians.

Since

1.0

5.28.2.6 Matrix() [6/6]

```
Leap::Matrix::Matrix (
            const Vector & axis,
            float angleRadians,
            const Vector & translation )  [inline]
```

Constructs a transformation matrix specifying a rotation around the specified vector and a translation by the specified vector.

Parameters

axis	A Vector specifying the axis of rotation.
angleRadians	The angle of rotation in radians.
translation	A Vector representing the translation part of the transform.

Since

1.0

5.28.3 Member Function Documentation

5.28.3.1 identity()

```
static const Matrix& Leap::Matrix::identity ( )  [inline], [static]
```

Returns the identity matrix specifying no translation, rotation, and scale.

Returns

The identity matrix.

Since

1.0

5.28.3.2 operator"!=()

```
bool Leap::Matrix::operator!= (
             const Matrix & other ) const  [inline]
```

Compare Matrix inequality component-wise.

Since

1.0

5.28.3.3 operator∗()

```
Matrix Leap::Matrix::operator* (
             const Matrix & other ) const  [inline]
```

Multiply transform matrices.

Combines two transformations into a single equivalent transformation.

Parameters

other	A Matrix to multiply on the right hand side.

Returns

A new Matrix representing the transformation equivalent to applying the other transformation followed by this transformation.

Since

1.0

5.28.3.4 operator*=()

```
Matrix& Leap::Matrix::operator*= (
            const Matrix & other )  [inline]
```

Multiply transform matrices and assign the product.

Since

1.0

5.28.3.5 operator==()

```
bool Leap::Matrix::operator== (
            const Matrix & other ) const  [inline]
```

Compare Matrix equality component-wise.

Since

1.0

5.28.3.6 rigidInverse()

```
Matrix Leap::Matrix::rigidInverse ( ) const  [inline]
```

Performs a matrix inverse if the matrix consists entirely of rigid transformations (translations and rotations). If the matrix is not rigid, this operation will not represent an inverse.

Note that all matrices that are directly returned by the API are rigid.

Returns

The rigid inverse of the matrix.

Since

1.0

5.28.3.7 setRotation()

```
void Leap::Matrix::setRotation (
             const Vector & axis,
             float angleRadians )  [inline]
```

Sets this transformation matrix to represent a rotation around the specified vector.

This function erases any previous rotation and scale transforms applied to this matrix, but does not affect translation.

Parameters

axis	A Vector specifying the axis of rotation.
angleRadians	The amount of rotation in radians.

Since

1.0

5.28.3.8 toArray3x3() [1/2]

```
template<typename T >
T* Leap::Matrix::toArray3x3 (
             T * output ) const  [inline]
```

Writes the 3x3 Matrix object to a 9 element row-major float or double array.

Translation factors are discarded.

Returns a pointer to the same data.

Since

1.0

5.28.3.9 toArray3x3() [2/2]

```
FloatArray Leap::Matrix::toArray3x3 ( ) const  [inline]
```

Convert a 3x3 Matrix object to a 9 element row-major float array.

Translation factors are discarded.

Returns a FloatArray struct to avoid dynamic memory allocation.

Since

1.0

5.28.3.10 toArray4x4() [1/2]

```
template<typename T >
T* Leap::Matrix::toArray4x4 (
             T * output ) const  [inline]
```

Writes the 4x4 Matrix object to a 16 element row-major float or double array.

Returns a pointer to the same data.

Since

1.0

5.28.3.11 toArray4x4() [2/2]

```
FloatArray Leap::Matrix::toArray4x4 ( ) const  [inline]
```

Convert a 4x4 Matrix object to a 16 element row-major float array.

Returns a FloatArray struct to avoid dynamic memory allocation.

Since

1.0

5.28.3.12 toMatrix3x3()

```
template<typename Matrix3x3Type >
const Matrix3x3Type Leap::Matrix::toMatrix3x3 ( ) const  [inline]
```

Convert a Leap::Matrix object to another 3x3 matrix type.

The new type must define a constructor function that takes each matrix element as a parameter in row-major order.

Translation factors are discarded.

Since

1.0

5.28.3.13 toMatrix4x4()

```
template<typename Matrix4x4Type >
const Matrix4x4Type Leap::Matrix::toMatrix4x4 ( ) const  [inline]
```

Convert a Leap::Matrix object to another 4x4 matrix type.

The new type must define a constructor function that takes each matrix element as a parameter in row-major order.

Since

1.0

5.28.3.14 toString()

```
std::string Leap::Matrix::toString ( ) const  [inline]
```

Write the matrix to a string in a human readable format.

Since

1.0

5.28.3.15 transformDirection()

```
Vector Leap::Matrix::transformDirection (
             const Vector & in ) const  [inline]
```

Transforms a vector with this matrix by transforming its rotation and scale only.

Parameters

in	The Vector to transform.

Returns

A new Vector representing the transformed original.

Since

1.0

5.28.3.16 transformPoint()

```
Vector Leap::Matrix::transformPoint (
             const Vector & in ) const  [inline]
```

Transforms a vector with this matrix by transforming its rotation, scale, and translation.

Translation is applied after rotation and scale.

Parameters

in	The Vector to transform.

Returns

A new Vector representing the transformed original.

Since

1.0

5.28.4 Friends And Related Function Documentation

5.28.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & out,
            const Matrix & matrix )  [friend]
```

Write the matrix to an output stream in a human readable format.

Since

1.0

5.28.5 Member Data Documentation

5.28.5.1 origin

```
Vector Leap::Matrix::origin
```

The translation factors for all three axes.

Since

1.0

5.28.5.2 xBasis

```
Vector Leap::Matrix::xBasis
```

The basis vector for the x-axis.

Since

1.0

5.28.5.3 yBasis

```
Vector Leap::Matrix::yBasis
```

The basis vector for the y-axis.

Since

1.0

5.28.5.4 zBasis

```
Vector Leap::Matrix::zBasis
```

The basis vector for the z-axis.

Since

1.0

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

- sample/include/LeapMath.h

5.29 Leap::Pointable Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Pointable:

Collaboration diagram for Leap::Pointable:

Public Types

- enum Zone { ZONE_NONE = 0, ZONE_HOVERING = 1, ZONE_TOUCHING = 2 }

Public Member Functions

- **Pointable** (PointableImplementation ∗)
- **Pointable** (FingerImplementation ∗)
- **Pointable** (ToolImplementation ∗)
- LEAP_EXPORT Pointable ()
- LEAP_EXPORT int32_t id () const
- LEAP_EXPORT Frame frame () const
- LEAP_EXPORT Hand hand () const
- LEAP_EXPORT Vector tipPosition () const
- LEAP_EXPORT Vector tipVelocity () const
- LEAP_EXPORT Vector direction () const
- LEAP_EXPORT float width () const
- LEAP_EXPORT float length () const
- LEAP_EXPORT bool isFinger () const
- LEAP_EXPORT bool isTool () const
- LEAP_EXPORT bool isExtended () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT Zone touchZone () const
- LEAP_EXPORT float touchDistance () const
- LEAP_EXPORT Vector stabilizedTipPosition () const
- LEAP_EXPORT float timeVisible () const
- LEAP_EXPORT bool operator== (const Pointable &) const
- LEAP_EXPORT bool operator!= (const Pointable &) const
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const Pointable & invalid ()

Friends

- LEAP_EXPORT friend std::ostream & operator<< (std::ostream &, const Pointable &)

Additional Inherited Members

5.29.1 Detailed Description

The Pointable class reports the physical characteristics of a detected finger or tool.

Both fingers and tools are classified as Pointable objects. Use the Pointable::isFinger() function to determine whether a Pointable object represents a finger. Use the Pointable::isTool() function to determine whether a Pointable object represents a tool. The Leap Motion software classifies a detected entity as a tool when it is thinner, straighter, and longer than a typical finger.

To provide touch emulation, the Leap Motion software associates a floating touch plane that adapts to the user's finger movement and hand posture. The Leap Motion interprets purposeful movements toward this plane as potential touch points. The Pointable class reports touch state with the touchZone and touchDistance values.

Note that Pointable objects can be invalid, which means that they do not contain valid tracking data and do not correspond to a physical entity. Invalid Pointable objects can be the result of asking for a Pointable object using an ID from an earlier frame when no Pointable objects with that ID exist in the current frame. A Pointable object created from the Pointable constructor is also invalid. Test for validity with the Pointable::isValid() function.

Since

1.0

5.29.2 Member Enumeration Documentation

5.29.2.1 Zone

`enum Leap::Pointable::Zone`

Defines the values for reporting the state of a Pointable object in relation to an adaptive touch plane.

Since

1.0

Enumerator

ZONE_NONE	The Pointable object is too far from the plane to be considered hovering or touching. **Since** 1.0
ZONE_HOVERING	The Pointable object is close to, but not touching the plane. **Since** 1.0
ZONE_TOUCHING	The Pointable has penetrated the plane. **Since** 1.0

5.29.3 Constructor & Destructor Documentation

5.29.3.1 Pointable()

```
LEAP_EXPORT Leap::Pointable::Pointable ( )
```

Constructs a Pointable object.

An uninitialized pointable is considered invalid. Get valid Pointable objects from a Frame or a Hand object.

Since

1.0

5.29.4 Member Function Documentation

5.29.4.1 direction()

```
LEAP_EXPORT Vector Leap::Pointable::direction ( ) const
```

The direction in which this finger or tool is pointing.

The direction is expressed as a unit vector pointing in the same direction as the tip.

Returns

The Vector pointing in the same direction as the tip of this Pointable object.

Since

1.0

5.29.4.2 frame()

```
LEAP_EXPORT Frame Leap::Pointable::frame ( ) const
```

The Frame associated with this Pointable object.

Returns

The associated Frame object, if available; otherwise, an invalid Frame object is returned.

Since

1.0

5.29.4.3 hand()

```
LEAP_EXPORT Hand Leap::Pointable::hand ( ) const
```

The Hand associated with a finger.

Not that in version 2+, tools are not associated with hands. For tools, this function always returns an invalid Hand object.

Returns

The associated Hand object, if available; otherwise, an invalid Hand object is returned.

Since

1.0

5.29.4.4 id()

```
LEAP_EXPORT int32_t Leap::Pointable::id ( ) const
```

A unique ID assigned to this Pointable object, whose value remains the same across consecutive frames while the tracked finger or tool remains visible. If tracking is lost (for example, when a finger is occluded by another finger or when it is withdrawn from the Leap Motion Controller field of view), the Leap Motion software may assign a new ID when it detects the entity in a future frame.

Use the ID value with the Frame::pointable() function to find this Pointable object in future frames.

IDs should be from 1 to 100 (inclusive). If more than 100 objects are tracked an IDs of -1 will be used until an ID in the defined range is available.

Returns

The ID assigned to this Pointable object.

Since

1.0

5.29.4.5 invalid()

```
static LEAP_EXPORT const Pointable& Leap::Pointable::invalid ( )  [static]
```

Returns an invalid Pointable object.

You can use the instance returned by this function in comparisons testing whether a given Pointable instance is valid or invalid. (You can also use the Pointable::isValid() function.)

Returns

The invalid Pointable instance.

Since

1.0

5.29.4.6 isExtended()

```
LEAP_EXPORT bool Leap::Pointable::isExtended ( ) const
```

Whether or not this Pointable is in an extended posture.

A finger is considered extended if it is extended straight from the hand as if pointing. A finger is not extended when it is bent down and curled towards the palm. Tools are always extended.

Returns

True, if the pointable is extended.

Since

2.0

5.29.4.7 isFinger()

```
LEAP_EXPORT bool Leap::Pointable::isFinger ( ) const
```

Whether or not this Pointable is classified as a finger.

Returns

True, if this Pointable is classified as a finger.

Since

1.0

5.29.4.8 isTool()

```
LEAP_EXPORT bool Leap::Pointable::isTool ( ) const
```

Whether or not this Pointable is classified as a tool.

Returns

True, if this Pointable is classified as a tool.

Since

1.0

5.29.4.9 isValid()

```
LEAP_EXPORT bool Leap::Pointable::isValid ( ) const
```

Reports whether this is a valid Pointable object.

Returns

True, if this Pointable object contains valid tracking data.

Since

1.0

5.29.4.10 length()

```
LEAP_EXPORT float Leap::Pointable::length ( ) const
```

The estimated length of the finger or tool in millimeters.

Returns

The estimated length of this Pointable object.

Since

1.0

5.29.4.11 operator"!=()

```
LEAP_EXPORT bool Leap::Pointable::operator!= (
             const Pointable &  ) const
```

Compare Pointable object inequality.

Two Pointable objects are equal if and only if both Pointable objects represent the exact same physical entities in the same frame and both Pointable objects are valid.

Since

1.0

5.29.4.12 operator==()

```
LEAP_EXPORT bool Leap::Pointable::operator== (
             const Pointable &  ) const
```

Compare Pointable object equality.

Two Pointable objects are equal if and only if both Pointable objects represent the exact same physical entities in the same frame and both Pointable objects are valid.

Since

1.0

5.29.4.13 stabilizedTipPosition()

```
LEAP_EXPORT Vector Leap::Pointable::stabilizedTipPosition ( ) const
```

The stabilized tip position of this Pointable.

Smoothing and stabilization is performed in order to make this value more suitable for interaction with 2D content. The stabilized position lags behind the tip position by a variable amount, depending primarily on the speed of movement.

Returns

A modified tip position of this Pointable object with some additional smoothing and stabilization applied.

Since

1.0

5.29.4.14 timeVisible()

```
LEAP_EXPORT float Leap::Pointable::timeVisible ( ) const
```

The duration of time this Pointable has been visible to the Leap Motion Controller.

Returns

The duration (in seconds) that this Pointable has been tracked.

Since

1.0

5.29.4.15 tipPosition()

```
LEAP_EXPORT Vector Leap::Pointable::tipPosition ( ) const
```

The tip position in millimeters from the Leap Motion origin.

Returns

The Vector containing the coordinates of the tip position.

Since

1.0

5.29.4.16 tipVelocity()

```
LEAP_EXPORT Vector Leap::Pointable::tipVelocity ( ) const
```

The rate of change of the tip position in millimeters/second.

Returns

The Vector containing the coordinates of the tip velocity.

Since

1.0

5.29.4.17 toString()

```
std::string Leap::Pointable::toString ( ) const  [inline]
```

A string containing a brief, human readable description of the Pointable object.

Returns

A description of the Pointable object as a string.

Since

1.0

5.29.4.18 touchDistance()

```
LEAP_EXPORT float Leap::Pointable::touchDistance ( ) const
```

A value proportional to the distance between this Pointable object and the adaptive touch plane.

The touch distance is a value in the range [-1, 1]. The value 1.0 indicates the Pointable is at the far edge of the hovering zone. The value 0 indicates the Pointable is just entering the touching zone. A value of -1.0 indicates the Pointable is firmly within the touching zone. Values in between are proportional to the distance from the plane. Thus, the touchDistance of 0.5 indicates that the Pointable is halfway into the hovering zone.

You can use the touchDistance value to modulate visual feedback given to the user as their fingers close in on a touch target, such as a button.

Returns

The normalized touch distance of this Pointable object.

Since

1.0

5.29.4.19 touchZone()

```
LEAP_EXPORT Zone Leap::Pointable::touchZone ( ) const
```

The current touch zone of this Pointable object.

The Leap Motion software computes the touch zone based on a floating touch plane that adapts to the user's finger movement and hand posture. The Leap Motion software interprets purposeful movements toward this plane as potential touch points. When a Pointable moves close to the adaptive touch plane, it enters the "hovering" zone. When a Pointable reaches or passes through the plane, it enters the "touching" zone.

The possible states are present in the Zone enum of this class:

Zone.NONE – The Pointable is outside the hovering zone.

Zone.HOVERING – The Pointable is close to, but not touching the touch plane.

Zone.TOUCHING – The Pointable has penetrated the touch plane.

The touchDistance value provides a normalized indication of the distance to the touch plane when the Pointable is in the hovering or touching zones.

Returns

The touch zone of this Pointable

Since

1.0

5.29.4.20 width()

```
LEAP_EXPORT float Leap::Pointable::width ( ) const
```

The estimated width of the finger or tool in millimeters.

Returns

The estimated width of this Pointable object.

Since

1.0

5.29.5 Friends And Related Function Documentation

5.29.5.1 operator<<

```
LEAP_EXPORT friend std::ostream& operator<< (
             std::ostream & ,
             const Pointable &   )  [friend]
```

Writes a brief, human readable description of the Pointable object to an output stream.

Since

1.0

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

- sample/include/Leap.h

5.30 Leap::PointableList Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::PointableList:

Collaboration diagram for Leap::PointableList:

Public Types

- typedef ConstListIterator< PointableList, Pointable > const_iterator

Public Member Functions

- **PointableList** (const ListBaseImplementation< Pointable > &)
- LEAP_EXPORT PointableList ()
- LEAP_EXPORT int count () const
- LEAP_EXPORT bool isEmpty () const
- LEAP_EXPORT Pointable operator[ ] (int index) const
- LEAP_EXPORT PointableList & append (const PointableList &other)
- LEAP_EXPORT PointableList & append (const FingerList &other)
- LEAP_EXPORT PointableList & append (const ToolList &other)
- LEAP_EXPORT Pointable leftmost () const
- LEAP_EXPORT Pointable rightmost () const
- LEAP_EXPORT Pointable frontmost () const
- LEAP_EXPORT PointableList extended () const
- LEAP_EXPORT const_iterator begin () const
- LEAP_EXPORT const_iterator end () const

Additional Inherited Members

5.30.1 Detailed Description

The PointableList class represents a list of Pointable objects.

Pointable objects include entities that can be pointed, such as fingers and tools.

Get a PointableList object by calling Frame::pointables() or Hand::pointables().

Since

1.0

5.30.2 Member Typedef Documentation

5.30.2.1 const_iterator

```
typedef ConstListIterator<PointableList, Pointable> Leap::PointableList::const_iterator
```

A C++ iterator type for PointableList objects.

Since

1.0

5.30.3 Constructor & Destructor Documentation

5.30.3.1 PointableList()

```
LEAP_EXPORT Leap::PointableList::PointableList ( )
```

Constructs an empty list of pointable entities.

Since

1.0

5.30.4 Member Function Documentation

5.30.4.1 append() [1/3]

```
LEAP_EXPORT PointableList& Leap::PointableList::append (
             const PointableList & other )
```

Appends the members of the specified PointableList to this PointableList.

Parameters

other	A PointableList object containing Pointable objects to append to the end of this PointableList.

Since

1.0

5.30.4.2 append() [2/3]

```
LEAP_EXPORT PointableList& Leap::PointableList::append (
             const FingerList & other )
```

Appends the members of the specified FingerList to this PointableList.

Parameters

other	A FingerList object containing Finger objects to append to the end of this PointableList.

Since

1.0

5.30.4.3 append() [3/3]

```
LEAP_EXPORT PointableList& Leap::PointableList::append (
             const ToolList & other )
```

Appends the members of the specified ToolList to this PointableList.

Parameters

other	A ToolList object containing Tool objects to append to the end of this PointableList.

Since

1.0

5.30.4.4 begin()

```
LEAP_EXPORT const_iterator Leap::PointableList::begin ( ) const
```

The C++ iterator set to the beginning of this PointableList.

Since

1.0

5.30.4.5 count()

```
LEAP_EXPORT int Leap::PointableList::count ( ) const
```

Returns the number of pointable entities in this list.

Returns

The number of pointable entities in this list.

Since

1.0

5.30.4.6 end()

```
LEAP_EXPORT const_iterator Leap::PointableList::end ( ) const
```

The C++ iterator set to the end of this PointableList.

Since

1.0

5.30.4.7 extended()

```
LEAP_EXPORT PointableList Leap::PointableList::extended ( ) const
```

Returns a new list containing those members of the current list that are extended. This includes all tools and any fingers whose isExtended() function is true.

Returns

The list of tools and extended fingers from the current list.

Since

2.0

5.30.4.8 frontmost()

```
LEAP_EXPORT Pointable Leap::PointableList::frontmost ( ) const
```

The member of the list that is farthest to the front within the standard Leap Motion frame of reference (i.e has the smallest Z coordinate).

Returns

The frontmost pointable, or invalid if list is empty.

Since

1.0

5.30.4.9 isEmpty()

```
LEAP_EXPORT bool Leap::PointableList::isEmpty ( ) const
```

Reports whether the list is empty.

Returns

True, if the list has no members.

Since

1.0

5.30.4.10 leftmost()

```
LEAP_EXPORT Pointable Leap::PointableList::leftmost ( ) const
```

The member of the list that is farthest to the left within the standard Leap Motion frame of reference (i.e has the smallest X coordinate).

Returns

The leftmost pointable, or invalid if list is empty.

Since

1.0

5.30.4.11 operator[]()

```
LEAP_EXPORT Pointable Leap::PointableList::operator[ ] (
              int index ) const
```

Access a list member by its position in the list.

Parameters

index	The zero-based list position index.

Returns

The Pointable object at the specified index.

Since

1.0

5.30.4.12 rightmost()

```
LEAP_EXPORT Pointable Leap::PointableList::rightmost ( ) const
```

The member of the list that is farthest to the right within the standard Leap Motion frame of reference (i.e has the largest X coordinate).

Returns

The rightmost pointable, or invalid if list is empty.

Since

1.0

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

- sample/include/Leap.h

5.31 SampleListener Class Reference

Inheritance diagram for SampleListener:

Collaboration diagram for SampleListener:

Public Member Functions

- virtual void onInit (const Controller &)
- virtual void onConnect (const Controller &)
- virtual void onDisconnect (const Controller &)
- virtual void onExit (const Controller &)
- virtual void onFrame (const Controller &)
- virtual void onFocusGained (const Controller &)
- virtual void onFocusLost (const Controller &)
- virtual void onDeviceChange (const Controller &)
- virtual void onServiceConnect (const Controller &)
- virtual void onServiceDisconnect (const Controller &)
- virtual void onInit (const Controller &)
- virtual void onConnect (const Controller &)
- virtual void onDisconnect (const Controller &)
- virtual void onExit (const Controller &)
- virtual void onFrame (const Controller &)
- virtual void onFocusGained (const Controller &)
- virtual void onFocusLost (const Controller &)
- virtual void onDeviceChange (const Controller &)
- virtual void onServiceConnect (const Controller &)
- virtual void onServiceDisconnect (const Controller &)

5.31.1 Member Function Documentation

5.31.1.1 onConnect() [1/2]

```
virtual void SampleListener::onConnect (
            const Controller &  )  [virtual]
```

Called when the Controller object connects to the Leap Motion software and the Leap Motion hardware device is plugged in, or when this Listener object is added to a Controller that is already connected.

When this callback is invoked, Controller::isServiceConnected is true, Controller::devices() is not empty, and, for at least one of the Device objects in the list, Device::isStreaming() is true.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.2 onConnect() [2/2]

```
void SampleListener::onConnect (
            const Controller &  )  [virtual]
```

Called when the Controller object connects to the Leap Motion software and the Leap Motion hardware device is plugged in, or when this Listener object is added to a Controller that is already connected.

When this callback is invoked, Controller::isServiceConnected is true, Controller::devices() is not empty, and, for at least one of the Device objects in the list, Device::isStreaming() is true.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.3 onDeviceChange() [1/2]

```
virtual void SampleListener::onDeviceChange (
            const Controller &  )  [virtual]
```

Called when a Leap Motion controller plugged in, unplugged, or the device changes state.

State changes include changes in frame rate and entering or leaving "robust" mode. Note that there is currently no way to query whether a device is in robust mode. You can use Frame::currentFramerate() to get the framerate.

Parameters

controller	The Controller object invoking this callback function.

Since

1.2

Reimplemented from Leap::Listener.

5.31.1.4 onDeviceChange() [2/2]

```
void SampleListener::onDeviceChange (
             const Controller &  )  [virtual]
```

Called when a Leap Motion controller plugged in, unplugged, or the device changes state.

State changes include changes in frame rate and entering or leaving "robust" mode. Note that there is currently no way to query whether a device is in robust mode. You can use Frame::currentFramerate() to get the framerate.

Parameters

controller	The Controller object invoking this callback function.

Since

1.2

Reimplemented from Leap::Listener.

5.31.1.5 onDisconnect() [1/2]

```
virtual void SampleListener::onDisconnect (
             const Controller &  )  [virtual]
```

Called when the Controller object disconnects from the Leap Motion software or the Leap Motion hardware is unplugged. The controller can disconnect when the Leap Motion device is unplugged, the user shuts the Leap Motion software down, or the Leap Motion software encounters an unrecoverable error.

Note: When you launch a Leap-enabled application in a debugger, the Leap Motion library does not disconnect from the application. This is to allow you to step through code without losing the connection because of time outs.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.6 onDisconnect() [2/2]

```
void SampleListener::onDisconnect (
             const Controller &  )  [virtual]
```

Called when the Controller object disconnects from the Leap Motion software or the Leap Motion hardware is unplugged. The controller can disconnect when the Leap Motion device is unplugged, the user shuts the Leap Motion software down, or the Leap Motion software encounters an unrecoverable error.

Note: When you launch a Leap-enabled application in a debugger, the Leap Motion library does not disconnect from the application. This is to allow you to step through code without losing the connection because of time outs.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.7 onExit() [1/2]

```
virtual void SampleListener::onExit (
             const Controller &  )  [virtual]
```

Called when this Listener object is removed from the Controller or the Controller instance is destroyed.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.8 onExit() [2/2]

```
void SampleListener::onExit (
             const Controller &  )  [virtual]
```

Called when this Listener object is removed from the Controller or the Controller instance is destroyed.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.9 onFocusGained() [1/2]

```
virtual void SampleListener::onFocusGained (
             const Controller &  )  [virtual]
```

Called when this application becomes the foreground application.

Only the foreground application receives tracking data from the Leap Motion Controller. This function is only called when the controller object is in a connected state.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.10 onFocusGained() [2/2]

```
void SampleListener::onFocusGained (
             const Controller &  )  [virtual]
```

Called when this application becomes the foreground application.

Only the foreground application receives tracking data from the Leap Motion Controller. This function is only called when the controller object is in a connected state.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.11 onFocusLost() [1/2]

```
virtual void SampleListener::onFocusLost (
            const Controller & )  [virtual]
```

Called when this application loses the foreground focus.

Only the foreground application receives tracking data from the Leap Motion Controller. This function is only called when the controller object is in a connected state.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.12 onFocusLost() [2/2]

```
void SampleListener::onFocusLost (
            const Controller & )  [virtual]
```

Called when this application loses the foreground focus.

Only the foreground application receives tracking data from the Leap Motion Controller. This function is only called when the controller object is in a connected state.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.13 onFrame() [1/2]

```
virtual void SampleListener::onFrame (
             const Controller & ) [virtual]
```

Called when a new frame of hand and finger tracking data is available. Access the new frame data using the Controller::frame() function.

Note, the Controller skips any pending onFrame events while your onFrame handler executes. If your implementation takes too long to return, one or more frames can be skipped. The Controller still inserts the skipped frames into the frame history. You can access recent frames by setting the history parameter when calling the Controller↩ ::frame() function. You can determine if any pending onFrame events were skipped by comparing the ID of the most recent frame with the ID of the last received frame.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.14 onFrame() [2/2]

```
void SampleListener::onFrame (
             const Controller & ) [virtual]
```

Called when a new frame of hand and finger tracking data is available. Access the new frame data using the Controller::frame() function.

Note, the Controller skips any pending onFrame events while your onFrame handler executes. If your implementation takes too long to return, one or more frames can be skipped. The Controller still inserts the skipped frames into the frame history. You can access recent frames by setting the history parameter when calling the Controller↩ ::frame() function. You can determine if any pending onFrame events were skipped by comparing the ID of the most recent frame with the ID of the last received frame.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.15 onInit() [1/2]

```
virtual void SampleListener::onInit (
             const Controller & )  [virtual]
```

Called once, when this Listener object is newly added to a Controller.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.16 onInit() [2/2]

```
void SampleListener::onInit (
             const Controller & )  [virtual]
```

Called once, when this Listener object is newly added to a Controller.

Parameters

controller	The Controller object invoking this callback function.

Since

1.0

Reimplemented from Leap::Listener.

5.31.1.17 onServiceConnect() [1/2]

```
virtual void SampleListener::onServiceConnect (
             const Controller &  )  [virtual]
```

Called when the Leap Motion daemon/service connects to your application Controller.

Parameters

controller	The Controller object invoking this callback function.

Since

1.2

Reimplemented from Leap::Listener.

5.31.1.18 onServiceConnect() [2/2]

```
void SampleListener::onServiceConnect (
             const Controller &  )  [virtual]
```

Called when the Leap Motion daemon/service connects to your application Controller.

Parameters

controller	The Controller object invoking this callback function.

Since

1.2

Reimplemented from Leap::Listener.

5.31.1.19 onServiceDisconnect() [1/2]

```
virtual void SampleListener::onServiceDisconnect (
             const Controller &  )  [virtual]
```

Called if the Leap Motion daemon/service disconnects from your application Controller.

Normally, this callback is not invoked. It is only called if some external event or problem shuts down the service or otherwise interrupts the connection.

Parameters

controller	The Controller object invoking this callback function.

Since

1.2

Reimplemented from Leap::Listener.

5.31.1.20 onServiceDisconnect() [2/2]

```
void SampleListener::onServiceDisconnect (
             const Controller &  )  [virtual]
```

Called if the Leap Motion daemon/service disconnects from your application Controller.

Normally, this callback is not invoked. It is only called if some external event or problem shuts down the service or otherwise interrupts the connection.

Parameters

controller	The Controller object invoking this callback function.

Since

1.2

Reimplemented from Leap::Listener.

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

- sample/ComprobarGesto.cpp
- sample/GuardarGesto.cpp

5.32 Leap::Screen Class Reference

Inheritance diagram for Leap::Screen:

Collaboration diagram for Leap::Screen:

Public Member Functions

- **Screen** (ScreenImplementation ∗)
- LEAP_EXPORT int32_t **id** () const
- LEAP_EXPORT Vector **intersect** (const Pointable &pointable, bool normalize, float clampRatio=1.0f) const
- LEAP_EXPORT Vector **intersect** (const Vector &position, const Vector &direction, bool normalize, float clampRatio=1.0f) const
- LEAP_EXPORT Vector **project** (const Vector &position, bool normalize, float clampRatio=1.0f) const
- LEAP_EXPORT Vector **horizontalAxis** () const
- LEAP_EXPORT Vector **verticalAxis** () const
- LEAP_EXPORT Vector **bottomLeftCorner** () const
- LEAP_EXPORT Vector **normal** () const
- LEAP_EXPORT int **widthPixels** () const
- LEAP_EXPORT int **heightPixels** () const
- LEAP_EXPORT float **distanceToPoint** (const Vector &point) const
- LEAP_EXPORT bool **isValid** () const
- LEAP_EXPORT bool **operator==** (const Screen &) const
- LEAP_EXPORT bool **operator!=** (const Screen &) const
- std::string **toString** () const

Static Public Member Functions

- static LEAP_EXPORT const Screen & **invalid** ()

Friends

- LEAP_EXPORT friend std::ostream & **operator<<** (std::ostream &, const Screen &)

Additional Inherited Members

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

- sample/include/Leap.h

5.33 Leap::ScreenList Class Reference

Inheritance diagram for Leap::ScreenList:

Collaboration diagram for Leap::ScreenList:

Public Types

- typedef ConstListIterator< ScreenList, Screen > **const_iterator**

Public Member Functions

- **ScreenList** (const ListBaseImplementation< Screen > &)
- LEAP_EXPORT int **count** () const
- LEAP_EXPORT bool **isEmpty** () const
- LEAP_EXPORT Screen **operator[ ]** (int index) const
- LEAP_EXPORT const_iterator **begin** () const
- LEAP_EXPORT const_iterator **end** () const
- LEAP_EXPORT Screen **closestScreenHit** (const Pointable &pointable) const
- LEAP_EXPORT Screen **closestScreenHit** (const Vector &position, const Vector &direction) const
- LEAP_EXPORT Screen **closestScreen** (const Vector &position) const

Additional Inherited Members

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

- sample/include/Leap.h

5.34 Leap::ScreenTapGesture Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::ScreenTapGesture:

Collaboration diagram for Leap::ScreenTapGesture:

Public Member Functions

- LEAP_EXPORT ScreenTapGesture ()
- LEAP_EXPORT ScreenTapGesture (const Gesture &rhs)
- LEAP_EXPORT Vector position () const
- LEAP_EXPORT Vector direction () const
- LEAP_EXPORT float progress () const
- LEAP_EXPORT Pointable pointable () const

Static Public Member Functions

- static Type classType ()

Additional Inherited Members

5.34.1 Detailed Description

The ScreenTapGesture class represents a tapping gesture by a finger or tool.

A screen tap gesture is recognized when the tip of a finger pokes forward and then springs back to approximately the original position, as if tapping a vertical screen. The tapping finger must pause briefly before beginning the tap.

Important: To use screen tap gestures in your application, you must enable recognition of the screen tap gesture. You can enable recognition with:

ScreenTap gestures are discrete. The ScreenTapGesture object representing a tap always has the state, STATE←↩ _STOP. Only one ScreenTapGesture object is created for each screen tap gesture recognized.

You can set the minimum finger movement and velocity required for a movement to be recognized as a screen tap as well as adjust the detection window for evaluating the movement using the config attribute of a connected Controller object. Use the following keys to configure screen tap recognition:

==================================== ========== ============= ======= Key string Value type Default value Units ==================================== ========== ============= ======= Gesture.ScreenTap.MinForwardVelocity float 50 mm/s Gesture.ScreenTap.HistorySeconds float 0.←↩ 1 s Gesture.ScreenTap.MinDistance float 5.0 mm ==================================== ========== ============= =======

The following example demonstrates how to set the screen tap configuration parameters:

The Controller object must be connected to the Leap Motion service/daemon before setting the configuration parameters.

Since

1.0

5.34.2 Constructor & Destructor Documentation

5.34.2.1 ScreenTapGesture() [1/2]

```
LEAP_EXPORT Leap::ScreenTapGesture::ScreenTapGesture ( )
```

Constructs a new ScreenTapGesture object.

An uninitialized ScreenTapGesture object is considered invalid. Get valid instances of the ScreenTapGesture class from a Frame object.

Since

1.0

5.34.2.2 ScreenTapGesture() [2/2]

```
LEAP_EXPORT Leap::ScreenTapGesture::ScreenTapGesture (
            const Gesture & rhs )
```

Constructs a ScreenTapGesture object from an instance of the Gesture class.

Parameters

rhs	The Gesture instance to specialize. This Gesture instance must be a ScreenTapGesture object.

Since

1.0

5.34.3 Member Function Documentation

5.34.3.1 classType()

```
static Type Leap::ScreenTapGesture::classType ( ) [inline], [static]
```

The screen tap gesture type.

Returns

Type The type value designating a screen tap gesture.

Since

1.0

5.34.3.2 direction()

```
LEAP_EXPORT Vector Leap::ScreenTapGesture::direction ( ) const
```

The direction of finger tip motion.

Returns

Vector A unit direction vector.

Since

1.0

5.34.3.3 pointable()

```
LEAP_EXPORT Pointable Leap::ScreenTapGesture::pointable ( ) const
```

The finger performing the screen tap gesture.

Returns

Pointable A Pointable object representing the tapping finger.

Since

1.0

5.34.3.4 position()

```
LEAP_EXPORT Vector Leap::ScreenTapGesture::position ( ) const
```

The position where the screen tap is registered.

Returns

Vector A Vector containing the coordinates of screen tap location.

Since

1.0

5.34.3.5 progress()

```
LEAP_EXPORT float Leap::ScreenTapGesture::progress ( ) const
```

The progress value is always 1.0 for a screen tap gesture.

Returns

float The value 1.0.

Since

1.0

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

- sample/include/Leap.h

5.35 Leap::SwipeGesture Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::SwipeGesture:

Collaboration diagram for Leap::SwipeGesture:

Public Member Functions

- LEAP_EXPORT SwipeGesture (const Gesture &rhs)
- LEAP_EXPORT Vector startPosition () const
- LEAP_EXPORT Vector position () const
- LEAP_EXPORT Vector direction () const
- LEAP_EXPORT float speed () const
- LEAP_EXPORT Pointable pointable () const

Static Public Member Functions

- static Type classType ()

Additional Inherited Members

5.35.1 Detailed Description

The SwipeGesture class represents a swiping motion a finger or tool.

SwipeGesture objects are generated for each visible finger or tool. Swipe gestures are continuous; a gesture object with the same ID value will appear in each frame while the gesture continues.

Important: To use swipe gestures in your application, you must enable recognition of the swipe gesture. You can enable recognition with:

You can set the minimum length and velocity required for a movement to be recognized as a swipe using the config attribute of a connected Controller object. Use the following keys to configure swipe recognition:

==================================== ========== ============= ======= Key string Value type Default value Units ==================================== ========== ============= ======= Gesture.Swipe.MinLength float 150 mm Gesture.Swipe.MinVelocity float 1000 mm/s ==================================== ========== ============= =======

The following example demonstrates how to set the swipe configuration parameters:

The Controller object must be connected to the Leap Motion service/daemon before setting the configuration parameters.

Since

1.0

5.35.2 Constructor & Destructor Documentation

5.35.2.1 SwipeGesture()

```
LEAP_EXPORT Leap::SwipeGesture::SwipeGesture (
            const Gesture & rhs )
```

Constructs a SwipeGesture object from an instance of the Gesture class.

Parameters

rhs	The Gesture instance to specialize. This Gesture instance must be a SwipeGesture object.

Since

1.0

5.35.3 Member Function Documentation

5.35.3.1 classType()

```
static Type Leap::SwipeGesture::classType ( )  [inline], [static]
```

The swipe gesture type.

Returns

Type The type value designating a swipe gesture.

Since

1.0

5.35.3.2 direction()

```
LEAP_EXPORT Vector Leap::SwipeGesture::direction ( ) const
```

The unit direction vector parallel to the swipe motion.

You can compare the components of the vector to classify the swipe as appropriate for your application. For example, if you are using swipes for two dimensional scrolling, you can compare the x and y values to determine if the swipe is primarily horizontal or vertical.

Returns

Vector The unit direction vector representing the swipe motion.

Since

1.0

5.35.3.3 pointable()

```
LEAP_EXPORT Pointable Leap::SwipeGesture::pointable ( ) const
```

The finger performing the swipe gesture.

Returns

Pointable A Pointable object representing the swiping finger.

Since

1.0

5.35.3.4 position()

```
LEAP_EXPORT Vector Leap::SwipeGesture::position ( ) const
```

The current position of the swipe.

Returns

Vector The current swipe position within the Leap Motion frame of reference, in mm.

Since

1.0

5.35.3.5 speed()

```
LEAP_EXPORT float Leap::SwipeGesture::speed ( ) const
```

The swipe speed in mm/second.

Returns

float The speed of the finger performing the swipe gesture in millimeters per second.

Since

1.0

5.35.3.6 startPosition()

```
LEAP_EXPORT Vector Leap::SwipeGesture::startPosition ( ) const
```

The position where the swipe began.

Returns

Vector The starting position within the Leap Motion frame of reference, in mm.

Since

1.0

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

- sample/include/Leap.h

5.36 Leap::Tool Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::Tool:

Collaboration diagram for Leap::Tool:

Public Member Functions

- **Tool** (ToolImplementation ∗)
- LEAP_EXPORT Tool ()
- LEAP_EXPORT Tool (const Pointable &)
- std::string toString () const

Static Public Member Functions

- static LEAP_EXPORT const Tool & invalid ()

Additional Inherited Members

5.36.1 Detailed Description

The Tool class represents a tracked tool.

Tools are Pointable objects that the Leap Motion software has classified as a tool.

Get valid Tool objects from a Frame object.

Note that Tool objects can be invalid, which means that they do not contain valid tracking data and do not correspond to a physical tool. Invalid Tool objects can be the result of asking for a Tool object using an ID from an earlier frame when no Tool objects with that ID exist in the current frame. A Tool object created from the Tool constructor is also invalid. Test for validity with the Tool::isValid() function.

Since

1.0

5.36.2 Constructor & Destructor Documentation

5.36.2.1 Tool() [1/2]

```
LEAP_EXPORT Leap::Tool::Tool ( )
```

Constructs a Tool object.

An uninitialized tool is considered invalid. Get valid Tool objects from a Frame object.

Since

1.0

5.36.2.2 Tool() [2/2]

```
LEAP_EXPORT Leap::Tool::Tool (
              const Pointable & )  [explicit]
```

If the specified Pointable object represents a tool, creates a copy of it as a Tool object; otherwise, creates an invalid Tool object.

Since

1.0

5.36.3 Member Function Documentation

5.36.3.1 invalid()

```
static LEAP_EXPORT const Tool& Leap::Tool::invalid ( )  [static]
```

Returns an invalid Tool object.

You can use the instance returned by this function in comparisons testing whether a given Tool instance is valid or invalid. (You can also use the Tool::isValid() function.)

Returns

The invalid Tool instance.

Since

1.0

5.36.3.2 toString()

```
std::string Leap::Tool::toString ( ) const  [inline]
```

A string containing a brief, human readable description of the Tool object.

Returns

A description of the Tool object as a string.

Since

1.0

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

- sample/include/Leap.h

5.37 Leap::ToolList Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::ToolList:

Collaboration diagram for Leap::ToolList:

Public Types

- typedef ConstListIterator< ToolList, Tool > const_iterator

Public Member Functions

- **ToolList** (const ListBaseImplementation< Tool > &)
- LEAP_EXPORT ToolList ()
- LEAP_EXPORT int count () const
- LEAP_EXPORT bool isEmpty () const
- LEAP_EXPORT Tool operator[ ] (int index) const
- LEAP_EXPORT ToolList & append (const ToolList &other)
- LEAP_EXPORT Tool leftmost () const
- LEAP_EXPORT Tool rightmost () const
- LEAP_EXPORT Tool frontmost () const
- LEAP_EXPORT const_iterator begin () const
- LEAP_EXPORT const_iterator end () const

Additional Inherited Members

5.37.1 Detailed Description

The ToolList class represents a list of Tool objects.

Get a ToolList object by calling Frame::tools().

Since

1.0

5.37.2 Member Typedef Documentation

5.37.2.1 const_iterator

```
typedef ConstListIterator<ToolList, Tool> Leap::ToolList::const_iterator
```

A C++ iterator type for ToolList objects.

Since

1.0

5.37.3 Constructor & Destructor Documentation

5.37.3.1 ToolList()

```
LEAP_EXPORT Leap::ToolList::ToolList ( )
```

Constructs an empty list of tools.

Since

1.0

5.37.4 Member Function Documentation

5.37.4.1 append()

```
LEAP_EXPORT ToolList& Leap::ToolList::append (
                const ToolList & other )
```

Appends the members of the specified ToolList to this ToolList.

Parameters

other	A ToolList object containing Tool objects to append to the end of this ToolList.

Since

1.0

5.37.4.2 begin()

```
LEAP_EXPORT const_iterator Leap::ToolList::begin ( ) const
```

The C++ iterator set to the beginning of this ToolList.

Since

1.0

5.37.4.3 count()

```
LEAP_EXPORT int Leap::ToolList::count ( ) const
```

Returns the number of tools in this list.

Returns

The number of tools in this list.

Since

1.0

5.37.4.4 end()

```
LEAP_EXPORT const_iterator Leap::ToolList::end ( ) const
```

The C++ iterator set to the end of this ToolList.

Since

1.0

5.37.4.5 frontmost()

```
LEAP_EXPORT Tool Leap::ToolList::frontmost ( ) const
```

The member of the list that is farthest to the front within the standard Leap Motion frame of reference (i.e has the smallest Z coordinate).

Returns

The frontmost tool, or invalid if list is empty.

Since

1.0

5.37.4.6 isEmpty()

```
LEAP_EXPORT bool Leap::ToolList::isEmpty ( ) const
```

Reports whether the list is empty.

Returns

True, if the list has no members.

Since

1.0

5.37.4.7 leftmost()

```
LEAP_EXPORT Tool Leap::ToolList::leftmost ( ) const
```

The member of the list that is farthest to the left within the standard Leap Motion frame of reference (i.e has the smallest X coordinate).

Returns

The leftmost tool, or invalid if list is empty.

Since

1.0

5.37.4.8 operator[]()

```
LEAP_EXPORT Tool Leap::ToolList::operator[ ] (
              int index ) const
```

Access a list member by its position in the list.

Parameters

index	The zero-based list position index.

Returns

The Tool object at the specified index.

Since

1.0

5.37.4.9 rightmost()

```
LEAP_EXPORT Tool Leap::ToolList::rightmost ( ) const
```

The member of the list that is farthest to the right within the standard Leap Motion frame of reference (i.e has the largest X coordinate).

Returns

The rightmost tool, or invalid if list is empty.

Since

1.0

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

- sample/include/Leap.h

5.38 Leap::TrackedQuad Class Reference

```
#include <Leap.h>
```

Inheritance diagram for Leap::TrackedQuad:

Collaboration diagram for Leap::TrackedQuad:

Public Member Functions

- **TrackedQuad** (TrackedQuadImplementation ∗)
- LEAP_EXPORT TrackedQuad ()
- LEAP_EXPORT float width () const
- LEAP_EXPORT float height () const
- LEAP_EXPORT int resolutionX () const
- LEAP_EXPORT int resolutionY () const
- LEAP_EXPORT bool visible () const
- LEAP_EXPORT Matrix orientation () const
- LEAP_EXPORT Vector position () const
- LEAP_EXPORT MaskList masks () const
- LEAP_EXPORT ImageList images () const
- LEAP_EXPORT bool isValid () const
- LEAP_EXPORT bool operator== (const TrackedQuad &) const
- LEAP_EXPORT bool operator!= (const TrackedQuad &) const
- std::string **toString** () const

Static Public Member Functions

- static LEAP_EXPORT const TrackedQuad & invalid ()

Friends

- LEAP_EXPORT friend std::ostream & operator<< (std::ostream &, const TrackedQuad &)

Additional Inherited Members

5.38.1 Detailed Description

Note: This class is an experimental API for internal use only. It may be removed without warning.

Represents a quad-like object tracked by the Leap Motion sensors.

Only one quad can be tracked. Once a supported quad is tracked, the state of that quad will be updated for each frame of Leap Motion tracking data.

A TrackedQuad object represents the state of the quad at one moment in time. Get a new object from subsequent frames to get the latest state information.

Since

2.2.6

5.38.2 Constructor & Destructor Documentation

5.38.2.1 TrackedQuad()

```
LEAP_EXPORT Leap::TrackedQuad::TrackedQuad ( )
```

Constructs a new TrackedQuad object. Do not use. Get valid TrackedQuads from a Controller or Frame object.

Since

2.2.6

5.38.3 Member Function Documentation

5.38.3.1 height()

```
LEAP_EXPORT float Leap::TrackedQuad::height ( ) const
```

The physical height of the quad display area in millimeters.

Since

2.2.6

5.38.3.2 images()

```
LEAP_EXPORT ImageList Leap::TrackedQuad::images ( ) const
```

The images from which the state of this TrackedQuad was derived. These are the same image objects that you can get from the Controller or Frame object from which you got this TrackedQuad.

Since

2.2.6

5.38.3.3 invalid()

```
static LEAP_EXPORT const TrackedQuad& Leap::TrackedQuad::invalid ( )  [static]
```

An invalid object.

Since

2.2.6

5.38.3.4 isValid()

```
LEAP_EXPORT bool Leap::TrackedQuad::isValid ( ) const
```

Reports whether this is a valid object.

Since

2.2.6

5.38.3.5 masks()

```
LEAP_EXPORT MaskList Leap::TrackedQuad::masks ( ) const
```

The list of masks for the current set of images. A mask is a bitmap indicating which pixels in the image contain fingers or part of the hand in front of the quad.

The mask at index 0 corresponds to the left image; that with index 1, to the right image.

Since

2.2.6

5.38.3.6 operator"!=()

```
LEAP_EXPORT bool Leap::TrackedQuad::operator!= (
            const TrackedQuad & ) const
```

Compares quad objects for inequality.

Since

2.2.6

5.38.3.7 operator==()

```
LEAP_EXPORT bool Leap::TrackedQuad::operator== (
            const TrackedQuad & ) const
```

Compares quad objects for equality.

Since

2.2.6

5.38.3.8 orientation()

```
LEAP_EXPORT Matrix Leap::TrackedQuad::orientation ( ) const
```

The orientation of the quad within the Leap Motion frame of reference.

Since

2.2.6

5.38.3.9 position()

```
LEAP_EXPORT Vector Leap::TrackedQuad::position ( ) const
```

The position of the center of the quad display area within the Leap Motion frame of reference. In millimeters.

Since

2.2.6

5.38.3.10 resolutionX()

```
LEAP_EXPORT int Leap::TrackedQuad::resolutionX ( ) const
```

The horizontal resolution of the quad display area in pixels. This value is set in a configuration file. It is not determined dynamically.

Since

2.2.6

5.38.3.11 resolutionY()

```
LEAP_EXPORT int Leap::TrackedQuad::resolutionY ( ) const
```

The vertical resolution of the quad display area in pixels. This value is set in a configuration file. It is not determined dynamically.

Since

2.2.6

5.38.3.12 visible()

```
LEAP_EXPORT bool Leap::TrackedQuad::visible ( ) const
```

Reports whether the quad is currently detected within the Leap Motion field of view.

Since

2.2.6

5.38.3.13 width()

```
LEAP_EXPORT float Leap::TrackedQuad::width ( ) const
```

The physical width of the quad display area in millimeters.

Since

2.2.6

5.38.4 Friends And Related Function Documentation

5.38.4.1 operator<<

```
LEAP_EXPORT friend std::ostream& operator<< (
             std::ostream & ,
             const TrackedQuad &  )  [friend]
```

Provides a brief, human-readable description of this quad.

Since

2.2.6

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

- sample/include/Leap.h

5.39 Leap::Vector Struct Reference

```
#include <LeapMath.h>
```

Public Member Functions

- Vector ()
- Vector (float _x, float _y, float _z)
- Vector (const Vector &vector)
- float magnitude () const
- float magnitudeSquared () const
- float distanceTo (const Vector &other) const
- float angleTo (const Vector &other) const
- float pitch () const
- float yaw () const
- float roll () const
- float dot (const Vector &other) const
- Vector cross (const Vector &other) const
- Vector normalized () const
- Vector operator- () const
- Vector operator+ (const Vector &other) const
- Vector operator- (const Vector &other) const
- Vector operator∗ (float scalar) const
- Vector operator/ (float scalar) const
- Vector & operator+= (const Vector &other)
- Vector & operator-= (const Vector &other)
- Vector & operator∗= (float scalar)
- Vector & operator/= (float scalar)
- std::string toString () const
- bool operator== (const Vector &other) const
- bool operator!= (const Vector &other) const
- bool isValid () const
- float operator[ ] (unsigned int index) const
- const float ∗ toFloatPointer () const
- template<typename Vector3Type >
  const Vector3Type toVector3 () const
- template<typename Vector4Type >
  const Vector4Type toVector4 (float w=0.0f) const

Static Public Member Functions

- static const Vector & zero ()
- static const Vector & xAxis ()
- static const Vector & yAxis ()
- static const Vector & zAxis ()
- static const Vector & left ()
- static const Vector & right ()
- static const Vector & down ()
- static const Vector & up ()
- static const Vector & forward ()
- static const Vector & backward ()

Public Attributes

- float x
- float y
- float z

Friends

- Vector operator$*$ (float scalar, const Vector &vector)
- std::ostream & operator$<<$ (std::ostream &out, const Vector &vector)

5.39.1 Detailed Description

The Vector struct represents a three-component mathematical vector or point such as a direction or position in three-dimensional space.

The Leap Motion software employs a right-handed Cartesian coordinate system. Values given are in units of real-world millimeters. The origin is centered at the center of the Leap Motion Controller. The x- and z-axes lie in the horizontal plane, with the x-axis running parallel to the long edge of the device. The y-axis is vertical, with positive values increasing upwards (in contrast to the downward orientation of most computer graphics coordinate systems). The z-axis has positive values increasing away from the computer screen.

Since

1.0

5.39.2 Constructor & Destructor Documentation

5.39.2.1 Vector() [1/3]

```
Leap::Vector::Vector ( )  [inline]
```

Creates a new Vector with all components set to zero.

Since

1.0

5.39.2.2 Vector() [2/3]

```
Leap::Vector::Vector (
            float _x,
            float _y,
            float _z )  [inline]
```

Creates a new Vector with the specified component values.

Since

1.0

5.39.2.3 Vector() [3/3]

```
Leap::Vector::Vector (
            const Vector & vector )  [inline]
```

Copies the specified Vector.

Since

1.0

5.39.3 Member Function Documentation

5.39.3.1 angleTo()

```
float Leap::Vector::angleTo (
            const Vector & other ) const  [inline]
```

The angle between this vector and the specified vector in radians.

The angle is measured in the plane formed by the two vectors. The angle returned is always the smaller of the two conjugate angles. Thus `A.angleTo(B) == B.angleTo(A)` and is always a positive value less than or equal to pi radians (180 degrees).

If either vector has zero length, then this function returns zero.

Parameters

other	A Vector object.

Returns

The angle between this vector and the specified vector in radians.

Since

1.0

5.39.3.2 backward()

```
static const Vector& Leap::Vector::backward ( )  [inline], [static]
```

The unit vector pointing backward along the positive z-axis: (0, 0, 1)

Since

1.0

5.39.3.3 cross()

```
Vector Leap::Vector::cross (
            const Vector & other ) const  [inline]
```

The cross product of this vector and the specified vector.

The cross product is a vector orthogonal to both original vectors. It has a magnitude equal to the area of a parallelogram having the two vectors as sides. The direction of the returned vector is determined by the right-hand rule. Thus `A.cross(B) == -B.cross(A).`

Parameters

other	A Vector object.

Returns

The cross product of this vector and the specified vector.

Since

1.0

5.39.3.4 distanceTo()

```
float Leap::Vector::distanceTo (
            const Vector & other ) const  [inline]
```

The distance between the point represented by this Vector object and a point represented by the specified Vector object.

Parameters

other	A Vector object.

Returns

The distance from this point to the specified point.

Since

1.0

5.39.3.5 dot()

```
float Leap::Vector::dot (
            const Vector & other ) const  [inline]
```

The dot product of this vector with another vector.

The dot product is the magnitude of the projection of this vector onto the specified vector.

Parameters

other	A Vector object.

Returns

The dot product of this vector and the specified vector.

Since

1.0

5.39.3.6 down()

```
static const Vector& Leap::Vector::down ( )  [inline], [static]
```

The unit vector pointing down along the negative y-axis: (0, -1, 0)

Since

1.0

5.39.3.7 forward()

```
static const Vector& Leap::Vector::forward ( )  [inline], [static]
```

The unit vector pointing forward along the negative z-axis: (0, 0, -1)

Since

1.0

5.39.3.8 isValid()

```
bool Leap::Vector::isValid ( ) const  [inline]
```

Returns true if all of the vector's components are finite. If any component is NaN or infinite, then this returns false.

Since

1.0

5.39.3.9 left()

```
static const Vector& Leap::Vector::left ( )  [inline], [static]
```

The unit vector pointing left along the negative x-axis: (-1, 0, 0)

Since

1.0

5.39.3.10 magnitude()

```
float Leap::Vector::magnitude ( ) const  [inline]
```

The magnitude, or length, of this vector.

The magnitude is the L2 norm, or Euclidean distance between the origin and the point represented by the (x, y, z) components of this Vector object.

Returns

The length of this vector.

Since

1.0

5.39.3.11 magnitudeSquared()

```
float Leap::Vector::magnitudeSquared ( ) const  [inline]
```

The square of the magnitude, or length, of this vector.

Returns

The square of the length of this vector.

Since

1.0

5.39.3.12 normalized()

```
Vector Leap::Vector::normalized ( ) const  [inline]
```

A normalized copy of this vector.

A normalized vector has the same direction as the original vector, but with a length of one.

Returns

A Vector object with a length of one, pointing in the same direction as this Vector object.

Since

1.0

5.39.3.13 operator"!=()

```
bool Leap::Vector::operator!= (
             const Vector & other ) const  [inline]
```

Compare Vector inequality component-wise.

Since

1.0

5.39.3.14 operator*()

```
Vector Leap::Vector::operator* (
             float scalar ) const  [inline]
```

Multiply vector by a scalar.

Since

1.0

5.39.3.15 operator*=()

```
Vector& Leap::Vector::operator*= (
             float scalar )  [inline]
```

Multiply vector by a scalar and assign the product.

Since

1.0

5.39.3.16 operator+()

```
Vector Leap::Vector::operator+ (
            const Vector & other ) const  [inline]
```

Add vectors component-wise.

Since

1.0

5.39.3.17 operator+=()

```
Vector& Leap::Vector::operator+= (
            const Vector & other )  [inline]
```

Add vectors component-wise and assign the sum.

Since

1.0

5.39.3.18 operator-() [1/2]

```
Vector Leap::Vector::operator- ( ) const  [inline]
```

A copy of this vector pointing in the opposite direction.

Returns

A Vector object with all components negated.

Since

1.0

5.39.3.19 operator-() [2/2]

```
Vector Leap::Vector::operator- (
             const Vector & other ) const  [inline]
```

Subtract vectors component-wise.

Since

1.0

5.39.3.20 operator-=()

```
Vector& Leap::Vector::operator-= (
             const Vector & other )  [inline]
```

Subtract vectors component-wise and assign the difference.

Since

1.0

5.39.3.21 operator/()

```
Vector Leap::Vector::operator/ (
             float scalar ) const  [inline]
```

Divide vector by a scalar.

Since

1.0

5.39.3.22 operator/=()

```
Vector& Leap::Vector::operator/= (
             float scalar )  [inline]
```

Divide vector by a scalar and assign the quotient.

Since

1.0

5.39.3.23 operator==()

```
bool Leap::Vector::operator== (
             const Vector & other ) const  [inline]
```

Compare Vector equality component-wise.

Since

1.0

5.39.3.24 operator[]()

```
float Leap::Vector::operator[] (
             unsigned int index ) const  [inline]
```

Index vector components numerically. Index 0 is x, index 1 is y, and index 2 is z.

Returns

The x, y, or z component of this Vector, if the specified index value is at least 0 and at most 2; otherwise, returns zero.

Since

1.0

5.39.3.25 pitch()

```
float Leap::Vector::pitch ( ) const  [inline]
```

The pitch angle in radians.

Pitch is the angle between the negative z-axis and the projection of the vector onto the y-z plane. In other words, pitch represents rotation around the x-axis. If the vector points upward, the returned angle is between 0 and pi radians (180 degrees); if it points downward, the angle is between 0 and -pi radians.

Returns

The angle of this vector above or below the horizon (x-z plane).

Since

1.0

5.39.3.26 right()

```
static const Vector& Leap::Vector::right ( )  [inline], [static]
```

The unit vector pointing right along the positive x-axis: (1, 0, 0)

Since

1.0

5.39.3.27 roll()

```
float Leap::Vector::roll ( ) const  [inline]
```

The roll angle in radians.

Roll is the angle between the y-axis and the projection of the vector onto the x-y plane. In other words, roll represents rotation around the z-axis. If the vector points to the left of the y-axis, then the returned angle is between 0 and pi radians (180 degrees); if it points to the right, the angle is between 0 and -pi radians.

Use this function to get roll angle of the plane to which this vector is a normal. For example, if this vector represents the normal to the palm, then this function returns the tilt or roll of the palm plane compared to the horizontal (x-z) plane.

Returns

The angle of this vector to the right or left of the y-axis.

Since

1.0

5.39.3.28 toFloatPointer()

```
const float* Leap::Vector::toFloatPointer ( ) const  [inline]
```

Cast the vector to a float array.

Since

1.0

5.39.3.29 toString()

```
std::string Leap::Vector::toString ( ) const  [inline]
```

Returns a string containing this vector in a human readable format: (x, y, z).

Since

1.0

5.39.3.30 toVector3()

```
template<typename Vector3Type >
const Vector3Type Leap::Vector::toVector3 ( ) const  [inline]
```

Convert a Leap::Vector to another 3-component Vector type.

The specified type must define a constructor that takes the x, y, and z components as separate parameters.

Since

1.0

5.39.3.31 toVector4()

```
template<typename Vector4Type >
const Vector4Type Leap::Vector::toVector4 (
             float w = 0.0f ) const  [inline]
```

Convert a Leap::Vector to another 4-component Vector type.

The specified type must define a constructor that takes the x, y, z, and w components as separate parameters. (The homogeneous coordinate, w, is set to zero by default, but you should typically set it to one for vectors representing a position.)

Since

1.0

5.39.3.32 up()

```
static const Vector& Leap::Vector::up ( ) [inline], [static]
```

The unit vector pointing up along the positive y-axis: (0, 1, 0)

Since

1.0

5.39.3.33 xAxis()

```
static const Vector& Leap::Vector::xAxis ( ) [inline], [static]
```

The x-axis unit vector: (1, 0, 0)

Since

1.0

5.39.3.34 yaw()

```
float Leap::Vector::yaw ( ) const [inline]
```

The yaw angle in radians.

Yaw is the angle between the negative z-axis and the projection of the vector onto the x-z plane. In other words, yaw represents rotation around the y-axis. If the vector points to the right of the negative z-axis, then the returned angle is between 0 and pi radians (180 degrees); if it points to the left, the angle is between 0 and -pi radians.

Returns

The angle of this vector to the right or left of the negative z-axis.

Since

1.0

5.39.3.35 yAxis()

```
static const Vector& Leap::Vector::yAxis ( )  [inline], [static]
```

The y-axis unit vector: (0, 1, 0)

Since

1.0

5.39.3.36 zAxis()

```
static const Vector& Leap::Vector::zAxis ( )  [inline], [static]
```

The z-axis unit vector: (0, 0, 1)

Since

1.0

5.39.3.37 zero()

```
static const Vector& Leap::Vector::zero ( )  [inline], [static]
```

The zero vector: (0, 0, 0)

Since

1.0

5.39.4 Friends And Related Function Documentation

5.39.4.1 operator*

```
Vector operator* (
            float scalar,
            const Vector & vector ) [friend]
```

Multiply vector by a scalar on the left-hand side (C++ only).

Since

1.0

5.39.4.2 operator<<

```
std::ostream& operator<< (
            std::ostream & out,
            const Vector & vector ) [friend]
```

Writes the vector to the output stream using a human readable format: (x, y, z).

Since

1.0

5.39.5 Member Data Documentation

5.39.5.1 x

```
float Leap::Vector::x
```

The horizontal component.

Since

1.0

5.39.5.2 y

```
float Leap::Vector::y
```

The vertical component.

Since

1.0

5.39.5.3 z

```
float Leap::Vector::z
```

The depth component.

Since

1.0

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

- sample/include/LeapMath.h

Chapter 6

File Documentation

6.1 main.cpp File Reference

Archivo pricipal, Proyecto final, Programacion bajo plataformas abiertas.

```
#include "./include/Includes.h"
```
Include dependency graph for main.cpp:

6.2 sample/ComprobarGesto.cpp File Reference

Archivo que permite comprobar cada frame del leap con el guardo en la base de datos, se toma como base el ejemplo dado en el sdk y se le realizan modificaciones.

```
#include <iostream>
#include <cstring>
#include "./include/Leap.h"
#include <fstream>
```
Include dependency graph for ComprobarGesto.cpp:

Classes

- class SampleListener

Macros

- #define **ANSI_COLOR_RESET** "\x1b[0m"
- #define **ANSI_COLOR_RED** "\x1b[31m"

Functions

- double **devolverX** (std::string dato)
- double **devolverY** (std::string dato)
- double **devolverZ** (std::string dato)
- int **ContarLineas** (std::string ruta)
- std::string **DevolverPrimeraLinea** (std::string ruta)
- std::string **DevolverQuieroLinea** (std::string ruta, int numeroLinea)
- double **CalcularPorcentaje** (double Teorico, double Experimental)
- double **PorcentajeError** (std::string vectorTeorico, std::string vectorExperimental)
- std::string **CompararVectores** (std::string vectorTeorico, std::string vectorExperimental)
- std::string **ComprobarActual** ()
- void **EscribirCadena** (std::string dato, int modo, std::string nombre)
- void **EscribirNumero** (int dato, int modo, std::string nombre)
- void **EscribirVector** (Leap::Vector dato, int modo, std::string nombre)
- int **ContarFrames** (std::string nombre)
- int **main** (int argc, char ∗∗argv)

Variables

- const std::string **fingerNames** [ ] = {"Thumb", "Index", "Middle", "Ring", "Pinky"}
- const std::string **boneNames** [ ] = {"Metacarpal", "Proximal", "Middle", "Distal"}
- const std::string **stateNames** [ ] = {"STATE_INVALID", "STATE_START", "STATE_UPDATE", "STATE_E←
ND"}

6.2.1 Detailed Description

Archivo que permite comprobar cada frame del leap con el guardo en la base de datos, se toma como base el ejemplo dado en el sdk y se le realizan modificaciones.

Author

Jesus Zuñiga Mendez

Version

1.0

Date

18 de julio de 2019

Copyright

Copyleft (l) 2019

6.3 sourcecode/tools.cpp File Reference

Archivo que contiene funciones utiles para el main.

```
#include "../include/Includes.h"
```
Include dependency graph for tools.cpp:

Functions

- void **EliminarGesto** (string gesto, string nombre)
- void **ImprimirRegistro** (std::string nombre)
- void **EscribirCadena** (std::string dato, int modo, std::string nombre)
- void **ActualizarRegistro** (std::string dato, int modo, std::string nombre)
- double **devolverX** (string dato)
- double **devolverY** (string dato)
- double **devolverZ** (string dato)
- void **CalcularGesto** ()
- int **ComprobarGestoManos** ()
- int **ComprobarGestoDedos** ()
- int **ComprobarGestoCantidad** ()
- int **GuardarGesto** (string nombre)
- int **BorrarGesto** (string nombre)
- int **MoverArchivos** (string nombre)
- string **QuitarEspacios** (string cadena)
- string **PonerEspacios** (string cadena)

6.3.1 Detailed Description

Archivo que contiene funciones utiles para el main.

Author

Jesus Zuñiga Mendez

Version

1.0

Date

18 de julio de 2019

Copyright

Copyleft (l) 2019

Index