Documentation

<https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#frame-motion>

[Download SDK](https://developer.leapmotion.com/downloads)Class List

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

[detail level 12345]

|  |  |
| --- | --- |
| \N**com** |  |
| \N**leapmotion** |  |
| \N**leap** |  |
| oC[**CircleGesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_circle_gesture.html) | The [**CircleGesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_circle_gesture.html) classes represents a circular finger movement |
| oC[**Config**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_config.html) | The [**Config**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_config.html) class provides access to Leap Motion system configuration information |
| |\C[**ValueType**](https://developer.leapmotion.com/documentation/Languages/Java/API/enumcom_1_1leapmotion_1_1leap_1_1_config_1_1_value_type.html) | Enumerates the possible data types for configuration values |
| oC[**Controller**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_controller.html) | The [**Controller**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_controller.html) class is your main interface to the Leap Motion [**Controller**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_controller.html) |
| |\C[**PolicyFlag**](https://developer.leapmotion.com/documentation/Languages/Java/API/enumcom_1_1leapmotion_1_1leap_1_1_controller_1_1_policy_flag.html) | The supported controller policies |
| oC[**Device**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_device.html) | The [**Device**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_device.html) class represents a physically connected device |
| oC[**DeviceList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_device_list.html) | The [**DeviceList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_device_list.html) class represents a list of [**Device**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_device.html) objects |
| oC[**Finger**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_finger.html) | The [**Finger**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_finger.html) class represents a tracked finger |
| oC[**FingerList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_finger_list.html) | The [**FingerList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_finger_list.html) class represents a list of [**Finger**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_finger.html) objects |
| oC[**Frame**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_frame.html) | The [**Frame**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_frame.html) class represents a set of hand and finger tracking data detected in a single frame |
| oC[**Gesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_gesture.html) | The [**Gesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_gesture.html) class represents a recognized movement by the user |
| |oC[**State**](https://developer.leapmotion.com/documentation/Languages/Java/API/enumcom_1_1leapmotion_1_1leap_1_1_gesture_1_1_state.html) | The possible gesture states |
| |\C[**Type**](https://developer.leapmotion.com/documentation/Languages/Java/API/enumcom_1_1leapmotion_1_1leap_1_1_gesture_1_1_type.html) | The supported types of gestures |
| oC[**GestureList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_gesture_list.html) | The [**GestureList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_gesture_list.html) class represents a list of [**Gesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_gesture.html) objects |
| oC[**Hand**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_hand.html) | The [**Hand**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_hand.html) class reports the physical characteristics of a detected hand |
| oC[**HandList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_hand_list.html) | The [**HandList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_hand_list.html) class represents a list of [**Hand**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_hand.html) objects |
| oC[**InteractionBox**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_interaction_box.html) | The [**InteractionBox**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_interaction_box.html) class represents a box-shaped region completely within the field of view of the Leap Motion controller |
| oC[**KeyTapGesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_key_tap_gesture.html) | The [**KeyTapGesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_key_tap_gesture.html) class represents a tapping gesture by a finger or tool |
| oC[**Listener**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_listener.html) | The [**Listener**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_listener.html) class defines a set of callback functions that you can override in a subclass to respond to events dispatched by the [**Controller**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_controller.html) object |
| oC[**Matrix**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_matrix.html) | The [**Matrix**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_matrix.html) struct represents a transformation matrix |
| oC[**Pointable**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_pointable.html) | The [**Pointable**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_pointable.html) class reports the physical characteristics of a detected finger or tool |
| |\C[**Zone**](https://developer.leapmotion.com/documentation/Languages/Java/API/enumcom_1_1leapmotion_1_1leap_1_1_pointable_1_1_zone.html) | Defines the values for reporting the state of a [**Pointable**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_pointable.html) object in relation to an adaptive touch plane |
| oC[**PointableList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_pointable_list.html) | The [**PointableList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_pointable_list.html) class represents a list of [**Pointable**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_pointable.html) objects |
| oC[**ScreenTapGesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_screen_tap_gesture.html) | The [**ScreenTapGesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_screen_tap_gesture.html) class represents a tapping gesture by a finger or tool |
| oC[**SwipeGesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_swipe_gesture.html) | The [**SwipeGesture**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_swipe_gesture.html) class represents a swiping motion of a finger or tool |
| oC[**Tool**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_tool.html) | The [**Tool**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_tool.html) class represents a tracked tool |
| oC[**ToolList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_tool_list.html) | The [**ToolList**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_tool_list.html) class represents a list of [**Tool**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_tool.html) objects |
| \C[**Vector**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_vector.html) | The [**Vector**](https://developer.leapmotion.com/documentation/Languages/Java/API/classcom_1_1leapmotion_1_1leap_1_1_vector.html) struct represents a three-component mathematical vector or point such as a direction or position in three-dimensional space |

# Getting Frame Data

The Leap Motion API presents motion tracking data to your application as a series of snapshots called frames. Each frame of tracking data contains the measured positions and other information about each entity detected in that snapshot. This article discusses the details of gettting Frame objects from the Leap Motion controller.

**Topics:**

* [Overview](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Frames.html#overview)
* [Getting Frames by Polling](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Frames.html#getting-frames-by-polling)
* [Getting Frames with Callbacks](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Frames.html#getting-frames-with-callbacks)
* [Getting Data from a Frame](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Frames.html#getting-data-from-a-frame)
* [Using IDs to track entities across frames](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Frames.html#using-ids-to-track-entities-across-frames)

# [Overview](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Frames.html#id1)

Get a Frame object containing tracking data from a connected Controller object. You can get a frame whenever your application is ready to process it using the frame() method of the Controller class:

if(controller.isConnected()) //controller is a Controller object

{

Frame frame = controller.frame(); //The latest frame

Frame previous = controller.frame(1); //The previous frame

}

The frame() function takes a history parameter that indicates how many frames back to retrieve. Typically, the last 60 frames are maintained in the history buffer.

# [Getting Frames by Polling](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Frames.html#id2)

Polling the Controller object for frames is the simplest and often best strategy when your application has a natural frame rate. You just call the Controller frame() function when your application is ready to process a frame of data.

When you use polling, there is a chance that you will get the same frame twice in a row (if the application frame rate exceeds the Leap frame rate) or skip a frame (if the Leap frame rate exceeds the application frame rate). In many cases, missed or duplicated frames are not important. For example, if you are moving an object on the screen in response to hand movement, the movement should still be smooth (assuming the overall frame rate of your application is high enough).

To detect whether you have already processed a frame, save the ID value assigned to the last frame processed and compare it to the current frame:

long lastFrameID = 0;

void processFrame(Frame frame )

{

if( frame.id() == lastFrameID ) return;

//...

lastFrameID = frame.id();

}

If your application has skipped frames, you can use the history parameter of the frame() function to access the skipped frames (as long as the Frame object is still in the history buffer):

int lastProcessedFrameID = 0;

void nextFrame( Controller controller )

{

long currentID = controller.frame().id();

for( int history = 0; history < currentID - lastFrameID; history++)

{

processFrame( controller.frame(history) );

}

lastFrameID = currentID;

}

void processNextFrame(Frame frame )

{

if( frame.isValid() )

{

//...

}

}

# [Getting Frames with Callbacks](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Frames.html#id3)

Alternatively, you can use a Listener object to get frames at the Leap Motion controller's frame rate. The Controller object calls the Listener's onFrame() function when a new frame is available. In the onFrame handler, you can call the Controller frame() function to get the Frame object itself.

Using the Listener callbacks is more complex because the callbacks are multi-threaded; each callback is invoked on an independent thread. You must ensure that any data accessed by multiple threads is handled in a thread-safe manner.

The following example defines a minimal Listener subclass that handles new frames of data:

class FrameListener extends Listener

{

public void onFrame(Controller controller)

{

Frame frame = controller.frame(); //The latest frame

Frame previous = controller.frame(1); //The previous frame

//...

}

};

As you can see, getting the tracking data through a Listener object is otherwise the same as polling the controller.

Note that it is possible to skip a frame even when using Listener callbacks. If your onFrame callback function takes too long to complete, then the next frame is added to the history, but the onFrame callback is skipped. Less commonly, if the Leap software itself cannot finish processing a frame in time, that frame can be abandoned and not added to the history. This problem can occur when a computer is bogged down with too many other computing tasks.

# [Getting Data from a Frame](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Frames.html#id4)

The Frame class defines several functions that provide access to the data in the frame. For example, the following code illustrates how to get the basic objects tracked by the Leap Motion system:

Controller controller = new Controller();

// wait until Controller.isConnected() evaluates to true

//...

Frame frame = controller.frame();

HandList hands = frame.hands();

PointableList pointables = frame.pointables();

FingerList fingers = frame.fingers();

ToolList tools = frame.tools();

The objects returned by the Frame object are all read-only. You can safely store them and use them in the future. They are thread-safe. Internally, the objects use the [C++ Boost library shared pointer class](http://www.boost.org/doc/libs/1_51_0/libs/smart_ptr/shared_ptr.htm#ThreadSafety).

# [Using IDs to track entities across frames](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Frames.html#id5)

If you have an ID of an entity from a different frame, you can get the object representing that entity in the current frame. Pass the ID to the Frame function of the appropriate type:

Hand hand = frame.hand(handID);

Pointable pointable = frame.pointable(pointableID);

Finger finger = frame.finger(fingerID);

Tool tool = frame.tool(toolID);

If an object with the same ID cannot be found -- perhaps a hand or finger moved out of the Leap field of view -- then a special, invalid object is returned instead. Invalid objects are instances of the appropriate class, but all their members return 0 values, zero vectors, or other invalid objects. This technique makes it more convenient to chain method calls together. For example, the following code snippet averages finger tip positions over several frames:

//Average a finger position for the last 10 frames

int count = 0;

Vector average = new Vector();

Finger fingerToAverage = frame.fingers().get(0);

for( int i = 0; i < 10; i++ )

{

Finger fingerFromFrame = controller.frame(i).finger(fingerToAverage.id());

if( fingerFromFrame.isValid() )

{

average = average.plus(fingerFromFrame.tipPosition());

count++;

}

}

average = average.divide(count);

Without invalid objects, this code would have to check each Frame object before checking the returned Finger objects.

# Leap Overview

The Leap detects and tracks hands, fingers and finger-like tools. The device operates in an intimate proximity with high precision and tracking frame rate.

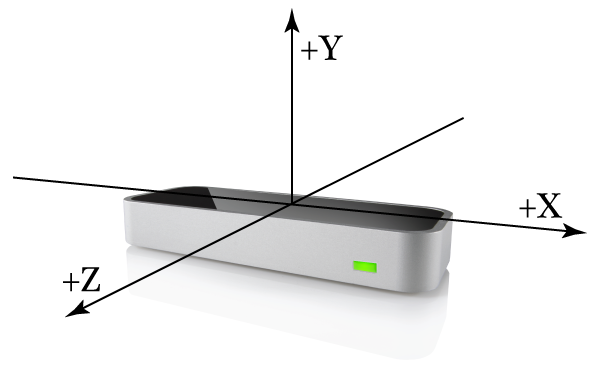
The Leap software analyzes the objects observed in the device field of view. It recognizes hands, fingers, and tools, reporting both discrete positions, gestures, and motion. The Leap field of view is an inverted pyramid centered on the device. The effective range of the Leap extends from approximately 25 to 600 millimeters above the device (1 inch to 2 feet).

**Topics:**

* [Coordinate system](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#coordinate-system)
* [Motion tracking data](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#motion-tracking-data)
  + [Frames](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#frames)
    - [Lists of tracking data](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#lists-of-tracking-data)
    - [Frame motion](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#frame-motion)
  + [Hand model](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#hand-model)
    - [Hand attributes](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#hand-attributes)
    - [Hand motion](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#hand-motion)
    - [Finger and Tool lists](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#finger-and-tool-lists)
  + [Finger and Tool models](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#finger-and-tool-models)
  + [Gestures](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#gestures)
    - [Circle](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#circle)
    - [Swipe](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#swipe)
    - [Taps](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#taps)
  + [Key Taps](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#key-taps)
  + [Screen Taps](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#screen-taps)

# [Coordinate system](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id1)

The Leap employs a right-handed Cartesian coordinate system. Values reported 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.



***The Leap right-handed coordinate system.***

# [Motion tracking data](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id2)

As the Leap tracks hands, fingers, and tools in its field of view, it provides updates as a set, or frame, of data. Each frame contains lists of the basic tracking data, such as hands, fingers, and tools, as well as recognized gestures and factors describing the overall motion in the scene.

When it detects a hand, finger, tool, or gesture, the Leap assigns it a unique ID designator. The ID remains the same as long as that entity remains visible within the device's field of view. If tracking is lost and regained, the Leap may assign a new ID (the software may not know that the hand or finger is the same as the one visible earlier).

Note: We plan to enhance the motion tracking data provided to your application before the Leap is released to consumers. In future releases of the Leap SDK, we plan to introduce a skeletal hand model to provide more detailed tracking data and continuity through time.

## [Frames](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id3)

A Frame object provides lists of the tracking data, gestures, and factors describing the overall motion observed in the Leap field of view.

### [Lists of tracking data](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id4)

* Hands — All hands.
* Pointables — All fingers and tools as Pointable objects.
* Fingers — All the fingers.
* Tools — All the tools.
* Gestures — All the gestures that started, ended, or which had an update.

The three pointables lists (Pointables, Fingers, and Tools) contain every pointable object detected in a frame. You can access the pointables associated with a hand through the Hand objects in the list of hands. Note that a finger or tool may not be associated with a hand if the user's hand is only partially within the Leap field of view.

If you are tracking an individual object, such as a finger, from frame to frame, you can use the ID associated with that object to look it up in each new frame. Use the following functions to look up specific object types by ID:

* Frame.hand()
* Frame.finger()
* Frame.tool()
* Frame.pointable()
* Frame.gesture()

These functions return a reference to the corresponding object if it exists in the current frame. If the object no longer exists, then a special invalid object is returned. Invalid objects are well-defined, but do not contain valid tracking data. This technique helps reduce the amount of null checking you have to do to when accessing Leap tracking data.

### [Frame motion](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id5)

The Leap analyzes the overall motion which occurred since an earlier frame and synthesizes representative translation, rotation, and scale factors. For example, if you move both hands to the left in the Leap field of view the frame contains translation. If you twist your hands as if turning a ball, the frame contains rotation. If you move your hands towards or away from each other, the frame contains scaling. The Leap uses all of the objects within the field of view when analyzing motion. If it only detects one hand, then the Leap bases the frame motion factors on the movement of that hand. If it detects two hands, then the Leap bases the frame motion factors on the movement of both hands together. You can also get independent motion factors for each hand from a Hand object.

Frame motions are derived by comparing the current frame with a specified earlier frame. The attributes describing the synthesized motion include:

* rotationAxis — A direction vector expressing the axis of rotation.
* rotationAngle — The angle of rotation clockwise around the rotation axis (using the right-hand rule).
* rotationMatrix — A transform matrix expressing the rotation.
* scaleFactor — A factor expressing expansion or contraction.
* translation — A vector expressing the linear movement.

You can apply the motion factors to manipulate objects in your application's scene without having to track individual hands and fingers over multiple frames.

## [Hand model](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id6)

The hand model provides information about the position, characteristics, and movement of a detected hand as well as lists of the fingers and tools associated with the hand.

The Leap API provides as much information about a hand as possible. However, the Leap may not be able to determine all hand attributes in every frame. For example, when a hand is clenched into a fist, its fingers are not visible to the Leap so the finger list will be empty. Your code should handle the cases where an attribute in the model is not available.

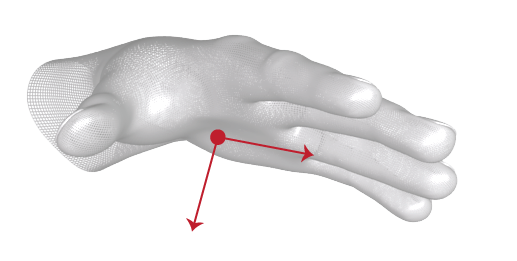
The Leap does not determine whether a hand is a left or right hand. More than two hands can appear in the hand list for a frame if more than one person's hands or other hand-like objects are in view. However, we recommend keeping at most two hands in the Leap Motion Controller's field of view for optimal motion tracking quality.

### [Hand attributes](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id7)

The Hand object provides several attributes reporting the physical characteristics of a detected hand:

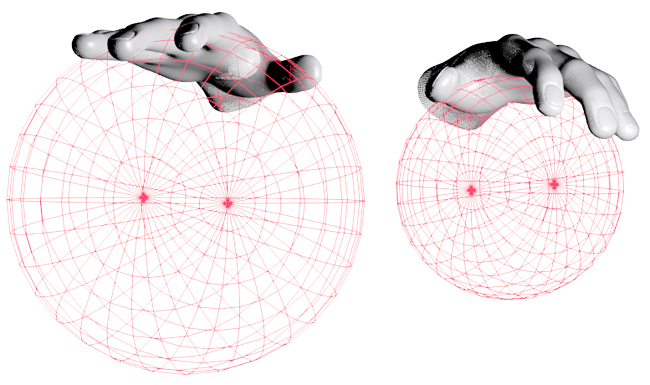
* palmPosition — The center of the palm measured in millimeters from the Leap origin.
* palmVelocity — The speed of the palm in millimeters per second.
* palmNormal — A vector perpendicular to the plane formed by the palm of the hand. The vector points downward out of the palm.
* direction — A vector pointing from the center of the palm toward the fingers.
* sphereCenter — The center of a sphere fit to the curvature of the hand (as if it were holding a ball).
* sphereRadius — The radius of a sphere fit to the curvature of the hand. The radius changes with the shape of the hand.

The direction and palmNormal are unit direction vectors describing the orientation of the hand with respect to the Leap coordinate system.



***The normal vector points perpendicularly out of the hand; the direction vector points forward.***

The sphereCenter and sphereRadius describe a sphere that is placed and sized to fit into the curvature of the hand:



***The size of the sphere decreases as the fingers are curled.***

### [Hand motion](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id8)

The Hand object also provides several attributes reporting the motion of a detected hand between frames. The Leap analyzes the motion of the hand, as well as its associated fingers and tools and reports representative translation, rotation, and scale factors. Moving your hand around the Leap field of view produces translation. Turning, twisting, or tilting your hand produces rotation. Moving fingers or tools toward or away from each other produces scaling.

Hand motions are derived by comparing the characteristics of the hand in the current frame to those in a specified earlier frame. The attributes describing the synthesized motion include:

* rotationAxis — A direction vector expressing the axis of rotation.
* rotationAngle — The angle of rotation clockwise around the rotation axis (using the right-hand rule).
* rotationMatrix — A transform matrix expressing the rotation.
* scaleFactor — A factor expressing expansion or contraction.
* translation — A vector expressing the linear movement.

### [Finger and Tool lists](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id9)

You can access the fingers and tools associated with a hand using one of three lists:

* Pointables — Both fingers and tools as Pointable objects.
* Fingers — Just the fingers.
* Tools — Just the tools.

You can also find an individual finger or tool using an ID value obtained in a previous frame. Use the Hand.finger(), the Hand.tool(), or, if you don't need to distinguish between fingers and tools, the Hand.pointable() function. These functions return a reference to the corresponding object in the current frame if it exists. If a finger or tool is not associated with the hand in this frame, then an invalid object is returned.

## [Finger and Tool models](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id10)

The Leap detects and tracks both fingers and tools within its field of view. The Leap classifies finger-like objects according to shape. A tool is longer, thinner, and straighter than a finger.

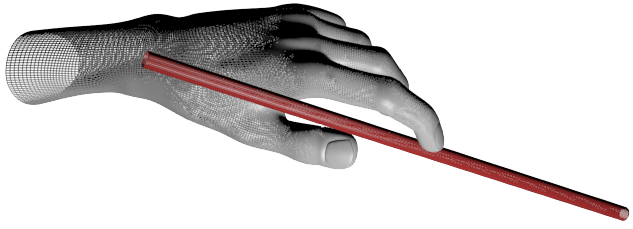
In the Leap model, the physical characteristics of fingers and tools are abstracted into a Pointable object. Fingers and tools are types of pointable objects. The physical characteristics of pointable objects include:

* length — The length of the visible portion of the object (from where it extends out of the hand to the tip).
* width — The average width of the visible portion of the object.
* direction — A unit direction vector pointing in the same direction as the object (i.e. from base to tip).
* tipPosition — The position of the tip in millimeters from the Leap origin.
* tipVelocity — The speed of the tip in millimeters per second.



***Finger tipPosition and direction vectors provide the positions of the finger tips and the directions in which the fingers are pointing.***

The Leap classifies a detected pointable object as either a finger or a tool. Use the Pointable.isTool() function to determine which one a Pointable object represents.



***A tool is longer, thinner, and straighter than a finger.***

## [Gestures](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id11)

The Leap recognizes certain movement patterns as gestures which could indicate a user intent or command. The Leap reports gestures observed in a frame the in the same way that it reports other motion tracking data like fingers and hands. For each gesture observed, the Leap adds a Gesture object to the frame. You can get these Gesture objects from the Frame gestures list.

 The following movement patterns are recognized by the Leap:

* Circle — A single finger tracing a circle.
* Swipe — A linear movement of the hand.
* Key Tap — A tapping movement by a finger as if tapping a keyboard key.
* Screen Tap — A tapping movement by the finger as if tapping a vertical computer screen.

When the Leap first classifies a movement pattern as a gesture, it adds a Gesture object to the frame. If the gesture continues over time, the Leap adds updated Gesture objects to subsequent frame. The gestures Circle and Swipe are continuous. The Leap updates the progress of these gestures each frame. Taps are discrete gestures. The Leap reports each tap with a single Gesture object.

**Important:** before using gestures in your application, you must enable recognition for each gesture you intend to use. The Controller class has an enableGesture() method that you can use to enable recognition for the types of gestures you use.

### [Circle](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id12)

The Leap recognizes the motion of a finger tracing a circle in space as a Circle gesture.



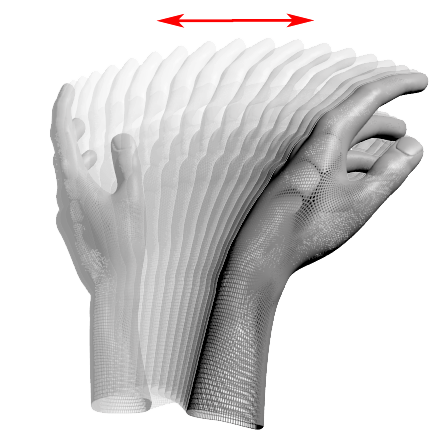
***A circle gesture with the forefinger.***

You can make a circle with any finger or tool. Circle gestures are continuous. Once the gesture starts, the Leap will update the progress until the gesture ends. A circle gesture ends when the circling finger or tool departs from the circle locus or moves too slow.

See CircleGesture in the API reference for more information.

### [Swipe](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id13)

The Leap recognizes a linear movement of a finger as a Swipe gesture.



***A horizontal swipe gesture.***

You can make a swipe gesture with any finger and in any direction. Swipe gestures are continuous. Once the gesture starts, the Leap will update the progress until the gesture ends. A swipe gesture ends when the finger changes directions or moves too slow.

See SwipeGesture in the API reference for more information.

### [Taps](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id14)

The Leap recognizes two types of taps: the downward Key Tap and the forward Screen Tap.

## [Key Taps](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id15)

The Leap recognizes a quick, downward tapping movement by a finger or tool as a Key Tap gesture.



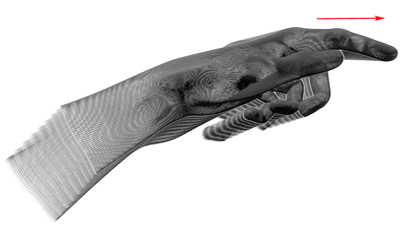
***A key tap gesture with the forefinger.***

You can make a key tap gesture by tapping downward as if pressing a piano key. Tap gestures are discrete. Only a single Gesture object is added per tap gesture.

See KeyTapGesture in the API reference for more information.

## [Screen Taps](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Overview.html#id16)

The Leap recognizes a quick, forward tapping movement by a finger or tool as a Screen Tap gesture.



***A screen tap gesture with the forefinger.***

You can make a key tap gesture by tapping or pushing foward in space as if touching a vertical touch screen. Tap gestures are discrete. Only a single Gesture object is added per tap gesture.

See ScreenTapGesture in the API reference for more information.

Copyright © 2012-2013 Leap Motion, Inc. All rights reserved.

Leap Motion proprietary and confidential. Not for distribution. Use subject to the terms of the Leap Motion SDK Agreement available at <https://developer.leapmotion.com/sdk_agreement>, or another agreement between Leap Motion and you, your company or other organization.

# Touch Emulation

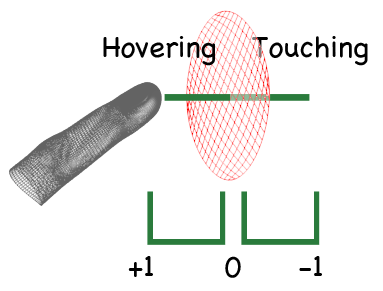
The Leap Motion API provides information that you can use to implement touch emulation in your application. Touch information is provided by the Pointable class.

**Topics:**

* [Overview](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#overview)
* [Getting the Touch Zone](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#getting-the-touch-zone)
* [Getting the Touch Distance](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#getting-the-touch-distance)
* [Getting the Stabilized Position of a Pointable](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#getting-the-stabilized-position-of-a-pointable)
* [Converting from Leap Motion Coordinates to Application Coordinates](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#converting-from-leap-motion-coordinates-to-application-coordinates)
* [TouchPoints Example](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#touchpoints-example)

# [Overview](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#id1)

The Leap defines an adaptive touch surface that you can use to orchestrate interaction with 2D elements of your application. This surface is oriented roughly parallel to the x-y plane, but adapts to the user's finger and hand position. As the user reaches forward with a finger or tool, the Leap reports whether that pointable object is close-to or touching this imaginary surface. The API reports touches with respect to the surface with two values: the zone and the distance to the touch plane.



***The virtual touch surface***

The touch zone identifies whether the Leap Motion software considers a Pointable as hovering near the touch surface, as penetrating the touch surface, or as too far from the surface (or pointing in the wrong direction). The zones are "hovering," "touching," and "none." The transition between zones tends to lag behind the touch distance. This lag is used to prevent abrupt and repeated transitions. If you are implementing touch interaction within an application, you may not need to consider the touch zone very often.

The touch distance is valid only when a Pointable is within the hovering or touching zones. The distance is a normalized value in the range [+1..-1]. When a Pointable first enters the hovering zone, the touch distance is +1.0 and the distance decreases toward 0 as the Pointable nears the touch surface. When the Pointable penetrates the surface, the distance is 0. As the Pointable pushes deeper into the touch zone, the distance approaches, but never exceeds, -1.

You can use the zone value to decide when to update UI elements based on hover or touch. You can use the distance to further modify UI elements based on proximity to the touch plane. For example, you can show the highlight state of a control when a finger is over the control and in the hovering zone and change the cursor based on distance to provide feedback about how close the user is to touching the control.

As part of the touch emulation API the Leap Motion provides a stabilized position for Pointable objects in addition to the standard position. The Leap Motion software stabilizes the position using an adaptive filter that smooths and slows the motion to make it easier for the user to interact with small regions on the screen (like buttons and links). The smoothing is greater when the movement is slow so that the user can zero in and touch a particular point more easily.

# [Getting the Touch Zone](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#id2)

The touch zone is reported by the touchZone attribute of the Pointable class. The zones are identified using the Zone enumeration, which defines the following states:

* NONE — the pointable is either too far from the touch surface to be considered touching, or it is pointing back toward the user.
* HOVERING — the pointable tip has crossed into the hovering zone, but isn't considered touching.
* TOUCHING — the pointable has crossed the virtual touch surface.

The following code snippet illustrates how to retrieve the zone of the forward-most finger:

Frame frame = leap.frame();

Pointable pointable = frame.pointables().frontmost();

Pointable.Zone zone = pointable.touchZone();

# [Getting the Touch Distance](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#id3)

The touch distance is reported by the touchDistance attribute of the Pointable class. The distance ranges from +1 to -1 as the finger moves to and though the virtual touch surface. The distance does not represent a physical quantity, but rather how close to touching the Leap Software considers the pointable.

The following code snippet illustrates how to retrieve the touch distance of the forward-most finger:

Frame frame = leap.frame();

Pointable pointable = frame.pointables().frontmost();

float distance = pointable.touchDistance();

# [Getting the Stabilized Position of a Pointable](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#id4)

The stabilized position is reported by the stabilizedTipPosition attribute of the Pointable class. This position is reported in reference to the standard Leap Motion coordinate system, but has a context-sensitive amount of filtering and stabilization.

The following code snippet illustrates how to retrieve the stabilized position of the forward-most finger:

Frame frame = leap.frame();

Pointable pointable = frame.pointables().frontmost();

Vector stabilizedPosition = pointable.stabilizedTipPosition();

# [Converting from Leap Motion Coordinates to Application Coordinates](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#id5)

When implementing touch emulation, you must map the Leap Motion coordinate space to the screen space of your application. To make this mapping easier, the Leap Motion API provides the InteractionBox class. The InteractionBox represents a rectilinear volume within the Leap Motion field of view. The class provides a function that normalizes positions within this volume to coordinates in the range [0..1]. You can normalize a position and then scale the resulting coordinate by the application dimensions to get a point in application coordinates.

For example, if you have a window with a client-area size represented by the variables windowWidth andwindowHeight, you can get the 2D pixel coordinates of a touch point within this window using the following code:

Frame frame = leap.frame();

Finger finger = frame.fingers().frontmost();

Vector stabilizedPosition = finger.stabilizedTipPosition();

InteractionBox iBox = leap.frame().interactionBox();

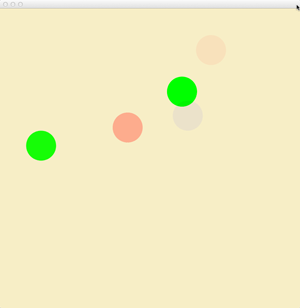
Vector normalizedPosition = iBox.normalizePoint(stabilizedPosition);

float x = normalizedPosition.getX() \* windowWidth;

float y = windowHeight - normalizedPosition.getY() \* windowHeight;

# [TouchPoints Example](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Touch_Emulation.html#id6)

The following example uses the touch emulation APIs to display the positions of all detected Pointable objects in an application window. The example uses the touch zone to set the color of the points and uses the touch distance to set the alpha value. The stabilized tip positions are mapped to the application window using the InteractionBox class.



***The TouchPoints example***

import com.leapmotion.leap.\*;

PShape cursor;

Controller leap;

int windowWidth = 800;

int windowHeight = 800;

color cursorNormalColor = color(0, 0, 128);

color cursorHoverColor = color(0, 128, 0);

color cursorTouchColor = color(128, 0, 0);

color backgroundColor = color(250, 235, 200);

void setup()

{

size(windowWidth, windowHeight, P2D);

leap = new Controller();

cursor = createShape(ELLIPSE, 0, 0, 40, 40);

cursor.stroke(cursorNormalColor);

cursor.fill(colorWithAlpha(cursorNormalColor, 16));

}

void draw()

{

background(backgroundColor);

InteractionBox iBox = leap.frame().interactionBox();

PointableList pointables = leap.frame().pointables();

for(int p = 0; p < pointables.count(); p++)

{

Pointable pointable = pointables.get(p);

Vector normalizedPosition = iBox.normalizePoint(pointable.stabilizedTipPosition());

float pixelX = normalizedPosition.getX() \* windowWidth;

float pixelY = windowHeight - normalizedPosition.getY() \* windowHeight;

int cx = (int) pixelX - 20;

int cy = (int) pixelY - 20;

Pointable.Zone fingerzone = pointable.touchZone();

if(pointable.touchDistance() > 0 && fingerzone != Pointable.Zone.ZONE\_NONE)

{

cursor.stroke(cursorHoverColor);

cursor.fill(colorWithAlpha(cursorHoverColor, (int)(64 - pointable.touchDistance() \* 48)));

}

else if(pointable.touchDistance() <= 0)

{

cursor.stroke(cursorTouchColor);

cursor.fill(colorWithAlpha(cursorTouchColor, (int)(64 - pointable.touchDistance() \* 126)));

}

else

{

cursor.stroke(cursorNormalColor);

cursor.fill(colorWithAlpha(cursorNormalColor, 16));

}

shape(cursor, cx, cy);

}

}

color colorWithAlpha(color rgb, int alpha)

{

return color(red(rgb), green(rgb), blue(rgb), alpha);

}

The example uses [Processing](http://www.processing.org/), but the Leap Motion-related code is applicable to all Java projects.

Copyright © 2012-2013 Leap Motion, Inc. All rights reserved.

Leap Motion proprietary and confidential. Not for distribution. Use subject to the terms of the Leap Motion SDK Agreement available at <https://developer.leapmotion.com/sdk_agreement>, or another agreement between Leap Motion and you, your company or other organization.

# Tracking Hands, Fingers, and Tools

Hands, fingers, and tools are the basic entities tracked by the Leap Motion system. This article discusses the details of gettting and using the objects representing these entities.

**Topics:**

* [Overview](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#overview)
* [Hand, and Pointable Lists](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#hand-and-pointable-lists)
* [Hands](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#hands)
  + [Getting the Hand Characteristics](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#getting-the-hand-characteristics)
  + [Getting the Fingers and Tools](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#getting-the-fingers-and-tools)
  + [Computing the Hand Orientation](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#computing-the-hand-orientation)
  + [Transforming Finger Coordinates into the Hand's Frame of Reference](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#transforming-finger-coordinates-into-the-hand-s-frame-of-reference)
* [Pointables](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#pointables)
  + [Converting a Pointable Object to a Finger or Tool](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#converting-a-pointable-object-to-a-finger-or-tool)
  + [Calculating the Position of the Base of a Finger](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#calculating-the-position-of-the-base-of-a-finger)

# [Overview](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#id1)

The Leap API defines a class representing each of the basic tracked objects.

The Frame object provides access to lists of hands, fingers, and tools. Fingers and tools are examples of pointable objects and can be treated together using the PointableList or separately using the FingerList and ToolList classes. Hand objects provide access to their fingers and tools (both together in a PointableList or separately).

The physical characteristics of hands, fingers, and tools are reported in reference to the Leap coordinate system (measured in units of millimeters). The Leap SDK provides a Vector class to describe points and directions. The Vector class provides several useful math functions for working with vectors.

# [Hand, and Pointable Lists](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#id2)

The list classes all have a similar structure. They are designed to act like vector-style arrays and support iterators. You cannot remove or alter the member objects of lists received from the Leap API, but you can combine lists of the same object type.

To use an iterator with one of these lists:

while(handList.iterator().hasNext())

{

Hand hand = handList.iterator().next();

System.out.println(hand);

}

The Hand, Pointable, Finger, and Tool lists define additional functions for getting members of the list based on their relative position within the Leap coordinate system. These functions include leftmost(), rightmost(), and frontmost(). The following snippet illustrates a few ways to use these functions:

Finger farLeft = frame.fingers().leftmost();

Finger mostForwardOnHand = frame.hands().get(0).fingers().frontmost();

Tool rightTool = frame.tools().rightmost();

A more complex example calculates the walls of a bounding box containing all detected pointable objects. This example defines its own functions for getting the top, bottom, and rear pointables since these are not included in the API:

float left = frame.pointables().leftmost().tipPosition().getX();

float right = frame.pointables().rightmost().tipPosition().getX();

float front = frame.pointables().frontmost().tipPosition().getZ();

float back = backmost(frame.pointables()).tipPosition().getZ();

float top = topmost(frame.pointables()).tipPosition().getY();

float bottom = bottommost(frame.pointables()).tipPosition().getY();

Pointable backmost(PointableList pointables)

{

if(pointables.count() == 0) return Pointable.invalid();

Pointable backmost = pointables.get(0);

for( int p = 1; p < pointables.count(); p++ )

{

if(pointables.get(p).tipPosition().getZ() > backmost.tipPosition().getZ())

backmost = pointables.get(p);

}

return backmost;

}

Pointable topmost(PointableList pointables)

{

if(pointables.count() == 0) return Pointable.invalid();

Pointable topmost = pointables.get(0);

for( int p = 1; p < pointables.count(); p++ )

{

if(pointables.get(p).tipPosition().getY() > topmost.tipPosition().getY())

topmost = pointables.get(p);

}

return topmost;

}

Pointable bottommost(PointableList pointables)

{

if(pointables.count() == 0) return Pointable.invalid();

Pointable bottommost = pointables.get(0);

for( int p = 1; p < pointables.count(); p++ )

{

if(pointables.get(p).tipPosition().getY() < bottommost.tipPosition().getY())

bottommost = pointables.get(p);

}

return bottommost;

}

# [Hands](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#id3)

The Hand class represents a physical hand detected by the Leap. A Hand object provides access to lists of its pointables as well as attributes describing the hand position, orientation, and movement. Get Hand objects from a Frame:

Frame frame = controller.frame(); // controller is a Controller object

HandList hands = frame.hands();

Hand firstHand = hands.get(0);

Or, if you know the ID from a previous frame:

Hand knownHand = frame.hand(handID);

You can also get hands by their relative positions in the frame:

Frame frame = controller.frame(); // controller is a Controller object

HandList hands = frame.hands();

Hand leftmost = hands.leftmost();

Hand rightmost = hands.rightmost();

Hand frontmost = hands.frontmost();

Note that the the leftmost() and rightmost() functions only identify which hand is farthest to the left or right. The functions do not identify which hand is the right or left hand.

## [Getting the Hand Characteristics](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#id4)

A hand is described by its position, orientation, and motion.

The hand's position is given by its palm position attribute, which provides a vector containing the 3-dimensional coordinates of the palm center point in millimeters from the Leap Motion origin. The hand's orientation is given by two vectors: the direction, which points from the palm center towards the fingers, and the palm normal, which points out of the palm, perpendicular to the plane of the hand.

The movement of the hand is given by the velocity attribute, which is a vector providing the instantaneous motion of the hand in mm/s. You can also get motion factors that translate how a hand has moved between two given frames into translation, rotation, and scaling values.

The following code snippet illustrates how to get a Hand object from a frame and access its basic attributes:

Hand hand = frame.hands().rightmost();

Vector position = hand.palmPosition();

Vector velocity = hand.palmVelocity();

Vector direction = hand.direction();

## [Getting the Fingers and Tools](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#id5)

You can get the fingers and tools associated with a hand as a list or individually using an ID obtained in a previous frame.

By list:

// hand is a Hand object

PointableList pointables = hand.pointables(); // Both fingers and tools

FingerList fingers = hand.fingers();

ToolList tools = hand.tools();

By ID from a previous frame:

Pointable knownPointable = hand.pointable(pointableID);

To get a finger or tool by relative position within the Leap field of view, use the right-, left- and frontmost functions of the matching list class:

// hand is a Hand object

Pointable leftPointable = hand.pointables().leftmost();

Finger rightFinger = hand.fingers().rightmost();

Tool frontTool = hand.tools().frontmost();

Note that these functions are relative to the Leap Motion origin, not to the hand itself. To get the fingers relative to the hand, you can use the Leap Matrix class to transform the finger positions into the hands frame of reference.

## [Computing the Hand Orientation](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#id6)

You can compute the hand orientation angles using the Hand direction and normal vectors. The Vector class defines functions for getting the pitch (angle around the x-axis), yaw (angle around the y-axis), and roll (angle around the z-axis):

float pitch = hand.direction().pitch();

float yaw = hand.direction().yaw();

float roll = hand.palmNormal().roll();

Note that the roll function only provides the expected angle when used with a normal vector.

## [Transforming Finger Coordinates into the Hand's Frame of Reference](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#id7)

Sometimes it is useful to obtain the coordinates of the fingers of a hand with respect to the hand's frame of reference. This lets you sort the fingers spatially and can simplify analysis of finger positions. You can create a transform matrix using the Leap Matrix class to transform the finger position and direction coordinates. The hand frame of reference can be usefully defined with the hand's direction and palm normal vectors, with the third axis defined by the cross product between the two. This puts the x-axis sideways across the hand, the z-axis pointing forward, and the y-axis parallel with the palm normal.

Frame frame = leap.frame();

for( int h = 0; h < frame.hands().count(); h++ )

{

Hand leapHand = frame.hands().get(h);

Vector handXBasis = leapHand.palmNormal().cross(leapHand.direction()).normalized();

Vector handYBasis = leapHand.palmNormal().opposite();

Vector handZBasis = leapHand.direction().opposite();

Vector handOrigin = leapHand.palmPosition();

Matrix handTransform = new Matrix(handXBasis, handYBasis, handZBasis, handOrigin);

handTransform = handTransform.rigidInverse();

for( int f = 0; f < leapHand.fingers().count(); f++ )

{

Finger leapFinger = leapHand.fingers().get(f);

Vector transformedPosition = handTransform.transformPoint(leapFinger.tipPosition());

Vector transformedDirection = handTransform.transformDirection(leapFinger.direction());

// Do something with the transformed fingers

}

}

# [Pointables](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#id8)

Pointable objects represent fingers and tools — i.e. things that can be pointed. You can get the fingers and tools associated with a particular hand from the Hand object. You can also get all detected pointables from a Frame object. Pointables are not necessarily associated with a Hand object — the physical hand itself may be out of the field of view or blocked by another hand. Thus the list of pointables from the frame can include fingers and tools not found in any list associated with a hand.

Pointable objects have many attributes describing the characteristics of the represented finger or tool:

* Tip position — the instantaneous position in mm from the Leap Motion origin.
* Tip velocity — the instantaneous velocity in mm/s.
* Stabilized tip position — the position filtered and stabilized using velocity and past positions.
* Direction — the current pointing direction vector.
* Length — the apparent length of the finger or tool.
* Width — the average width.
* Touch distance — the normalized distance from the virtual touch plane.
* Touch zone — the pointable's current relation to the virtual touch plane.

The following example illustrates how to get a pointable object from a frame and access its basic characteristics:

Pointable pointable = frame.pointables().frontmost();

Vector direction = pointable.direction();

float length = pointable.length();

float width = pointable.width();

Vector stabilizedPosition = pointable.stabilizedTipPosition();

Vector position = pointable.tipPosition();

Vector speed = pointable.tipVelocity();

float touchDistance = pointable.touchDistance();

Pointable.Zone zone = pointable.touchZone();

## [Converting a Pointable Object to a Finger or Tool](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#id9)

To convert a Pointable object to its proper Finger or Tool subclass, use the appropriate Finger or Tool constructor (one of the few times that you should use the constructor for a Leap class).

if (pointable.isTool()) {

Tool tool = new Tool(pointable);

} else {

Finger finger = new Finger(pointable);

}

## [Calculating the Position of the Base of a Finger](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Leap_Tracking.html#id10)

If you need to calculate the position of the base of the finger, you can use the finger tip position and direction as follows:

Vector basePosition = pointable.direction().opposite().times(pointable.length());

basePosition = basePosition.plus(pointable.tipPosition());

# Understanding the Java Sample Application

This article discusses the Java sample application included with the Leap SDK. After reading this article, you should be ready to access Leap hand tracking data from your own Java applications.

**Topics:**

* [Overview](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#overview)
* [Creating a Controller object](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#creating-a-controller-object)
* [Subclassing the Listener class](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#subclassing-the-listener-class)
* [Getting a Frame of data](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#getting-a-frame-of-data)
* [Getting Gestures](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#getting-gestures)
* [Running the sample](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#running-the-sample)

In the Leap SDK folder, you can find the following files used for this article:

Windows:

* LeapSDK/sample/Sample.java — Java sample application
* LeapSDK/lib/LeapJava.jar — Leap Java API class definitions
* LeapSDK/lib/x86/LeapJava.dll — 32-bit Leap Java library for Windows
* LeapSDK/lib/x64/LeapJava.dll — 64-bit Leap Java library for Windows
* LeapSDK/lib/x86/Leap.dll — 32-bit Leap library for Windows
* LeapSDK/lib/x64/Leap.dll — 64-bit Leap library for Windows
* LeapSDK/lib/x86/Leapd.dll — 32-bit Leap runtime debug library for Windows
* LeapSDK/lib/x64/Leapd.dll — 64-bit Leap runtime debug library for Windows

Mac:

* LeapSDK/sample/Sample.java — Java sample application
* LeapSDK/lib/LeapJava.jar — Leap Java API class definitions
* LeapSDK/lib/libLeapJava.dylib — Leap Java library for Mac
* LeapSDK/lib/libLeap.dylib — Leap library for Mac

# [Overview](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#id1)

In a nutshell, the Leap motion tracking device detects and tracks hands and fingers placed within its field of view. The Leap captures this data one frame at a time. Your applications can use the Leap API to access this data.

The sample application demonstrates how to use the Leap API to listen for frame events dispatched by the Leap and how to access the hand and finger data in each frame. The application is a small command-line program that prints information about detected hands and fingers to standard output. The application is contained in a single file,Sample.java.

The sample application uses most of the key classes in the Leap API, including:

* Controller — the interface between the Leap and your application
* Listener — used to handle events dispatched by the Leap
* Frame — contains a set of hand and finger tracking data
* Hand — contains tracking data for a detected hand
* Finger — contains tracking data for a detected finger
* Vector — represents a 3D position or directional vector
* Gesture — represents a recognized gesture.

For more detailed information about these classes, pleases refer to the Leap API reference documentation.

# [Creating a Controller object](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#id2)

The Controller class provides the main interface between the Leap and your application. When you create a Controller object, it connects to the Leap software running on the computer and makes hand tracking data available through Frame objects. You can access these Frame objects by instantiating a Controller object and calling theController.frame method.

If your application has a natural update loop or frame rate, then you can call Controller.frame as part of this update. Otherwise, you can add a listener to the controller object. The controller object invokes the callback methods defined in your Listener subclass whenever a new frame of tracking data is available (and also for a few other Leap events).

The sample application creates a Controller object in the Sample class's main method and adds an instance of a Listener subclass to the Controller with the Controller.addListener method:

class Sample {

public static void main(String[] args) {

// Create a sample listener and controller

SampleListener listener = new SampleListener();

Controller controller = new Controller();

// Have the sample listener receive events from the controller

controller.addListener(listener);

// Keep this process running until Enter is pressed

System.out.println("Press Enter to quit...");

try {

System.in.read();

} catch (IOException e) {

e.printStackTrace();

}

// Remove the sample listener when done

controller.removeListener(listener);

}

}

The application then runs until you press the Enter key. In the meantime, the controller calls the appropriate listener callback methods when events occur. Before running the program, though, you must create your own subclass of the Listener class.

# [Subclassing the Listener class](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#id3)

The sample application defines a Listener subclass, SampleListener, which implements callback methods to handle events dispatched by the Leap. The events include:

* onInit — dispatched once, when the controller to which the listener is registered is initialized.
* onConnect — dispatched when the controller connects to the Leap and is ready to begin sending frames of motion tracking data.
* onDisconnect — dispatched if the controller disconnects from the Leap (for example, if you unplug the Leap device or shut down the Leap software).
* onExit — dispatched to a listener when it is removed from a controller.
* onFrame — dispatched when a new frame of motion tracking data is available.

For three lifecycle event callbacks, onInit, onDisconnect, and onExit the sample application simply prints a message to standard output. For the onConnect and onFrame event, the listener callback does a bit more work. When the controller calls the onConnect callback, the function enables recognition for all the gesture types. When the controller calls onFrame, the function gets the latest frame of motion tracking data and prints information about the detected objects to standard output.

# [Getting a Frame of data](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#id4)

The Controller calls the onFrame callback method when the Leap generates a new frame of motion tracking data. You can access the new data by calling the Controller.frame method, which returns the newest Frame object. (A reference to the Controller object is passed to the callback as a parameter.) A Frame object contains an ID, a timestamp, and a list containing a Hand object for each physical hand in view.

The following code from the sample application's onFrame implementation gets the most recent Frame object from the controller, retrieves the list of Hands from the Frame and then prints out the Frame ID, timestamp, and the number of Hands in the list:

// Get the most recent frame and report some basic information

Frame frame = controller.frame();

System.out.println("Frame id: " + frame.id()

+ ", timestamp: " + frame.timestamp()

+ ", hands: " + frame.hands().count()

+ ", fingers: " + frame.fingers().count()

+ ", tools: " + frame.tools().count());

The method goes on to examine the first Hand in the list:

if (!frame.hands().isEmpty()) {

// Get the first hand

Hand hand = frame.hands().get(0);

A Hand object contains an ID, properties representing the hand's physical characteristics, and a list of Finger objects. Each Finger object contains an ID and properties representing the characteristic of the finger.

Once it has retrieved a hand, the method checks it for fingers and then averages the finger tip positions, printing the result and the number of fingers:

// Check if the hand has any fingers

FingerList fingers = hand.fingers();

if (!fingers.isEmpty()) {

// Calculate the hand's average finger tip position

Vector avgPos = Vector.zero();

for (Finger finger : fingers) {

avgPos = avgPos.plus(finger.tipPosition());

}

avgPos = avgPos.divide(fingers.count());

System.out.println("Hand has " + fingers.count()

+ " fingers, average finger tip position: " + avgPos);

}

Next, the method prints the radius of a sphere fit to the hand's curvature, along with the hand's palm position:

// Get the hand's sphere radius and palm position

System.out.println("Hand sphere radius: " + hand.sphereRadius()

+ " mm, palm position: " + hand.palmPosition());

Finally, the onFrame method uses Vector methods to calculate the hand's pitch, roll, and yaw angles from the hand's normal vector and the direction vector. The angles are converted from radians to degrees:

// Get the hand's normal vector and direction

Vector normal = hand.palmNormal();

Vector direction = hand.direction();

// Calculate the hand's pitch, roll, and yaw angles

System.out.println("Hand pitch: " + Math.toDegrees(direction.pitch()) + " degrees, "

+ "roll: " + Math.toDegrees(normal.roll()) + " degrees, "

+ "yaw: " + Math.toDegrees(direction.yaw()) + " degrees\n");

# [Getting Gestures](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#id5)

To receive gestures from the Leap, you first have to enable recognition for each type of gesture you are interested in. You can enable gesture recognition any time after the controller connects to the Leap (isConnected() is true). In the sample program, all gestures are enabled in the onConnect() callback function using the enableGesture() methods defined in the Controller class:

public void onConnect(Controller controller) {

System.out.println("Connected");

controller.enableGesture(Gesture.Type.TYPE\_SWIPE);

controller.enableGesture(Gesture.Type.TYPE\_CIRCLE);

controller.enableGesture(Gesture.Type.TYPE\_SCREEN\_TAP);

controller.enableGesture(Gesture.Type.TYPE\_KEY\_TAP);

}

The Leap adds Gesture objects representing each recognized movement pattern to the gestures list in the Frame object. In the onFrame() callback, the sample application loops through the gesture list and prints information about each one to the standard output. This operation is performed with a standard for-loop and switch statement.

The Gesture API uses a base Gesture class which is extended by classes representing the individual gestures. The objects in the gesture list are Gesture instances, so you must convert the Gesture instance to an instance of the correct subclass. Casting is not supported, instead each subclass provides a constructor that performs the conversion. For example, a Gesture instance representing a circle gesture can by converted into a CircleGesture instance with the following code:

CircleGesture circle = new CircleGesture(gesture);

If you try to convert a Gesture instance into the wrong subclass, the constructor functions return invalid Gesture objects.

It is sometimes useful to compare the properties of a gesture in the current frame to those from an earlier frame. For example, the circle gesture has a progress attribute that describes how many times the finger has traversed the circle. This is the total progress, however; if you want the progress between frames, you must subtract the progress value of the gesture in the previous frame. You can do this by looking up the gesture in the previous frame using the gesture ID. The following code calculates the progress since the previous frame to derive the angle in radians:

// Calculate angle swept since last frame

double sweptAngle = 0;

if (circle.state() != State.STATE\_START) {

CircleGesture previousUpdate = new CircleGesture(controller.frame(1).gesture(circle.id()));

sweptAngle = (circle.progress() - previousUpdate.progress()) \* 2 \* Math.PI;

}

The full code for the gesture loop is:

GestureList gestures = frame.gestures();

for (int i = 0; i < gestures.count(); i++) {

Gesture gesture = gestures.get(i);

switch (gesture.type()) {

case TYPE\_CIRCLE:

CircleGesture circle = new CircleGesture(gesture);

// Calculate clock direction using the angle between circle normal and pointable

String clockwiseness;

if (circle.pointable().direction().angleTo(circle.normal()) <= Math.PI/4) {

// Clockwise if angle is less than 90 degrees

clockwiseness = "clockwise";

} else {

clockwiseness = "counterclockwise";

}

// Calculate angle swept since last frame

double sweptAngle = 0;

if (circle.state() != State.STATE\_START) {

CircleGesture previousUpdate = new CircleGesture(controller.frame(1).gesture(circle.id()));

sweptAngle = (circle.progress() - previousUpdate.progress()) \* 2 \* Math.PI;

}

System.out.println("Circle id: " + circle.id()

+ ", " + circle.state()

+ ", progress: " + circle.progress()

+ ", radius: " + circle.radius()

+ ", angle: " + Math.toDegrees(sweptAngle)

+ ", " + clockwiseness);

break;

case TYPE\_SWIPE:

SwipeGesture swipe = new SwipeGesture(gesture);

System.out.println("Swipe id: " + swipe.id()

+ ", " + swipe.state()

+ ", position: " + swipe.position()

+ ", direction: " + swipe.direction()

+ ", speed: " + swipe.speed());

break;

case TYPE\_SCREEN\_TAP:

ScreenTapGesture screenTap = new ScreenTapGesture(gesture);

System.out.println("Screen Tap id: " + screenTap.id()

+ ", " + screenTap.state()

+ ", position: " + screenTap.position()

+ ", direction: " + screenTap.direction());

break;

case TYPE\_KEY\_TAP:

KeyTapGesture keyTap = new KeyTapGesture(gesture);

System.out.println("Key Tap id: " + keyTap.id()

+ ", " + keyTap.state()

+ ", position: " + keyTap.position()

+ ", direction: " + keyTap.direction());

break;

default:

System.out.println("Unknown gesture type.");

break;

}

}

# [Running the sample](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html#id6)

To run the sample application:

1. Compile the sample application:

* **On Windows, make sure that Sample.java and LeapJava.jar are in the current directory and run the following command in a command-line prompt:**

javac -classpath LeapJava.jar Sample.java

* **On Mac, make sure that Sample.java and LeapJava.jar are in the current directory and run the following command in a terminal window:**

javac -classpath ./LeapJava.jar Sample.java

1. Plug the Leap device into a USB port and place it in front of you.
2. If you haven't already, install the Leap software.
3. Start the Leap software. If prompted, enter your registered email address and password when prompted. The Leap icon appears in the notification area of the task bar (on Windows) or finder bar (on Mac) and turns green when ready.
4. Run the sample application:

* **On Windows, make sure that Sample.class, LeapJava.jar,LeapJava.dll, and Leap.dll are in the current directory. If you are using a 32-bit version of the Java Virtual Machine (JVM), use the .dll files from the lib\x86 folder of the SDK. Use the .dll files from lib\x64 with a 64-bit JVM. Run the following command in a command-line prompt:**

java -classpath "LeapJava.jar;." Sample

* **On Mac, make sure that Sample.class, LeapJava.jar,libLeapJava.dylib, and libLeap.dylib are in the current directory and run the following command in a terminal window:**

java -classpath "./LeapJava.jar:." Sample

You should see the messages "Initialized" and "Connected" printed to standard output when the application initializes and connects to the Leap. You should then see frame information printed each time the Leap dispatches the onFrameevent. When you place a hand above the Leap, you should also see finger and palm position information printed.

Now that you have seen how to access motion tracking data from the Leap, you can begin developing your own Java applications that integrate the Leap.

Copyright © 2012-2013 Leap Motion, Inc. All rights reserved.

Leap Motion proprietary and confidential. Not for distribution. Use subject to the terms of the Leap Motion SDK Agreement available at <https://developer.leapmotion.com/sdk_agreement>, or another agreement between Leap Motion and you, your company or other organization.

# Setting up a Java project

The Leap Motion Java SDK uses a standard Jar file for Leap Motion API class definitions and a set of native libraries that allow your Leap-enabled Java programs to exchange data with the Leap. Setting up a Java project typically involves adding the LeapJava.jar file to your application's classpath and setting the JVM library path parameter so that your JVM can find the native libraries.

**Topics:**

* [Leap Motion Java libraries](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#leap-motion-java-libraries)
* [Compile from the command line](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#compile-from-the-command-line)
* [Run from the command line](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#run-from-the-command-line)
* [Eclipse](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#eclipse)
* [IntelliJ](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#intellij)
* [NetBeans](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#netbeans)

# [Leap Motion Java libraries](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#id1)

The Leap Motion Jar file is cross-platform, but the native libraries must match the platform and architecture of the JVM used to run the program.

To use the Leap Motion SDK in a Java program, you must add the LeapJava.jar file to the classpath and set the java.library.path to the location of the Leap Motion native libraries.

Use the following Java and native libraries with the Leap Motion Java SDK:

* 32-bit Windows:
  + LeapSDK/lib/LeapJava.jar — Leap Motion Java API class definitions
  + LeapSDK/lib/x86/LeapJava.dll — 32-bit Leap Motion Java library for Windows
  + LeapSDK/lib/x86/Leap.dll — 32-bit Leap Motion library for Windows
* 64-bit Windows:
  + LeapSDK/lib/LeapJava.jar — Leap Motion Java API class definitions
  + LeapSDK/lib/x64/LeapJava.dll — 64-bit Leap Motion Java library for Windows
  + LeapSDK/lib/x64/Leap.dll — 64-bit Leap Motion library for Windows
* 32- or 64-bit Mac OS:
  + LeapSDK/lib/LeapJava.jar — Leap Motion Java API class definitions
  + LeapSDK/lib/libLeapJava.dylib — Leap Motion Java library for Mac
  + LeapSDK/lib/libLeap.dylib — Leap Motion library for Mac

# [Compile from the command line](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#id2)

Use the Java compiler, javac to compile, setting the classpath option to specify the LeapJar file. For example, to compile the Sample.java program included in the Leap Motion SDK, you could use the following command:

javac -classpath <LeapSDK>/lib/LeapJava.jar Sample.java

(where <LeapSDK> is the location of your Leap Motion SDK folder.)

# [Run from the command line](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#id3)

To launch a Leap-enabled program, Java needs to find the Leap Motion native libraries at runtime. LeapJava.jar must also be on the classpath. You can set Java's java.library.path parameter to identify the native library. The command line syntax is slightly different between Mac and Windows. More importantly, on Windows, you have to specify either the 32-bit or the 64-bit libraries to match the architecture of the JVM you are using.

On Mac, you could run the Sample program using the following command:

java -classpath ".:<LeapSDK>/lib/LeapJava.jar" -Djava.library.path=<LeapSDK>/lib Sample

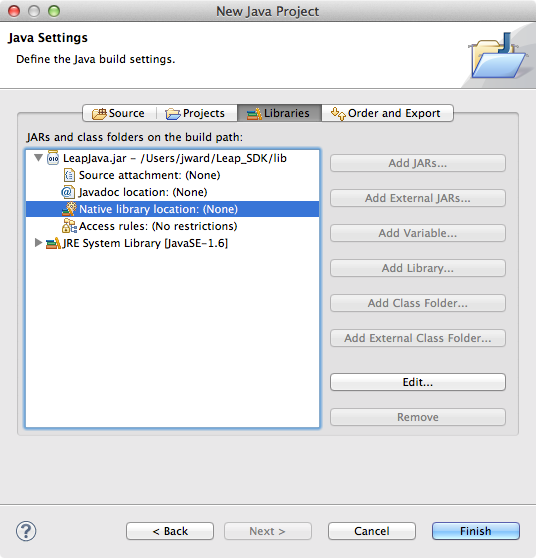
On Windows, you could run the Sample program using a 64-bit JVM with the following command:

java -classpath ".;<LeapSDK>/lib/LeapJava.jar" -Djava.library.path=<LeapSDK>/lib/x64 Sample

# [Eclipse](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#id4)

In the Eclipse IDE, you add the LeapJava.jar file to a project as an external Jar and then set the path to the appropriate native Leap Motion libraries as a property of the Jar file.

1. Select New > Java Project from the Eclipse File menu.
2. Assign a name to the project on the Create Java Project page and set other properties as desired. (The Leap Motion SDK supports Java 6 and 7.)
3. Click Next to advance to the Java Settings page.
4. Select the Libraries tab.
5. Click the Add External Jars... button.
6. Navigate to the LeapJava.jar file.
7. Click Open to add LeapJava.jar to the project.
8. Next, click the small triangle in front of the LeapJava.jar entry in the library list to reveal the library properties.



1. Select the Native library location item.
2. Click the Edit button.
3. Navigate to the folder containing the Leap Motion native libraries.

 On Windows, be sure to select the folder containing the correct libraries for your target architecture. If you are targeting a 32-bit JVM, use the Leap Motion libraries in the x86 folder of the SDK. If you are targeting a 64-bit JVM, use the libraries in the x64 folder. On Mac, the Leap Motion library files each support both architectures.

1. Click Ok to set the path.

Note: you can also add the Leap Motion libraries to an existing project from the Project Properties dialog.

# [IntelliJ](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#id5)

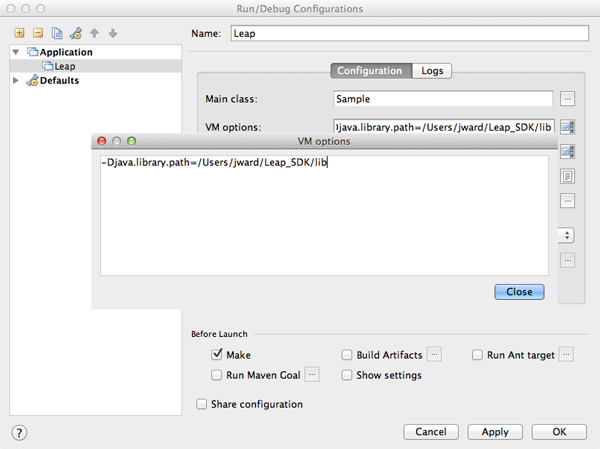
In the IntelliJ IDE, you add the LeapJava.jar file to a project as a library. You separately set the path to the Leap Motion native libraries using the JVM parameter, java.library.path. The JVM parameters can be set using an IntelliJ Run/Debug configuration.

To add LeapJava.jar to the project:

1. After creating a project in the usual way, select the File > Project Structure menu command to open the settings dialog.
2. Click Libraries under project settings.
3. Click the small + button at the top of the library list to open the Select Library Files dialog.
4. Add LeapJava.jar from your Leap Motion SDK.

To set the path to the native Leap Motion libraries by creating a Run/Debug configuration:

1. Select the Run > Edit Configurations… menu command.
2. Click the small + button above the Configuration list.
3. Choose Application to create a new application configuration.
4. Assign a name.
5. Set the VM Options field to set the -Djava.library.path parameter to the path to the proper folder containing the Leap Motion native libraries.
6. Click Ok.



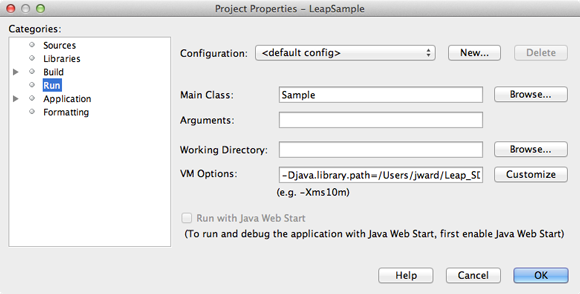
***On Windows, be sure to select the folder containing the correct libraries for your target architecture. If you are targeting a 32-bit JVM, use the Leap Motion libraries in the x86 folder of the SDK. If you are targeting a 64-bit JVM, use the libraries in the x64 folder. On Mac, the Leap Motion library files each support both architectures.***

# [NetBeans](https://developer.leapmotion.com/documentation/Languages/Java/Guides/Setup_Java.html#id6)

In the NetBeans IDE, you add the LeapJava.jar file to a project as a library. You separately set the path to the Leap Motion native libraries using the JVM parameter, java.library.path. The JVM parameters can be set using a NetBeans Run configuration.

To add the LeapJava.jar to a project and set the path to the native libraries:

1. After creating a project in the usual way, select the File > Project Properties menu command to open the Project Properties dialog.
2. Click the Libraries item.
3. Click Add Jar/Folder button.
4. Find the LeapJava.jar file in your Leap Motion SDK.
5. Click Ok to add the Jar file to your project.
6. Next, click the Run item in the Project properties list
7. Set the VM Options field to set the -Djava.library.path parameter to the path to the proper folder containing the Leap Motion native libraries.



***On Windows, be sure to select the folder containing the correct libraries for your target architecture. If you are targeting a 32-bit JVM, use the Leap Motion libraries in the x86 folder of the SDK. If you are targeting a 64-bit JVM, use the libraries in the x64 folder. On Mac, the Leap Motion library files each support both architectures.***

Copyright © 2012-2013 Leap Motion, Inc. All rights reserved.

Leap Motion proprietary and confidential. Not for distribution. Use subject to the terms of the Leap Motion SDK Agreement available at <https://developer.leapmotion.com/sdk_agreement>, or another agreement between Leap Motion and you, your company or other organization.