



## **USER MANUAL**

**Haptic SDK  
version 3.14.0**

Force Dimension  
Switzerland

[www.forcedimension.com](http://www.forcedimension.com)



<b>1 DHD - Haptic SDK Documentation</b>	<b>1</b>
1.1 Introduction	1
1.1.1 Multi-platforms	1
1.1.2 High-level vs. Low-level SDK	1
1.2 Features	2
1.2.1 Device Types	2
1.2.2 Axis Convention	2
1.2.3 Device Modes	3
1.2.4 Device Status	3
1.2.5 Support for Multiple Devices	4
1.2.6 Velocity Estimator	4
1.2.7 TimeGuard	4
1.2.8 Thread-Safe Operation	4
1.2.9 Multi-threading	5
1.2.10 Error Management	5
1.2.11 Safety Feature	5
1.2.12 Units	5
1.2.13 Standard SDK	6
1.2.14 Expert SDK	8
1.2.15 Controller SDK	10
1.2.16 OS-independent SDK	10
1.3 Technical Support	10
<b>2 GLOSSARY</b>	<b>11</b>
<b>3 Deprecated List</b>	<b>12</b>
<b>4 File Index</b>	<b>12</b>
4.1 File List	12
<b>5 File Documentation</b>	<b>12</b>
5.1 dhdc.h File Reference	12
5.1.1 Detailed Description	17
5.1.2 Macro Definition Documentation	17
5.1.3 Enumeration Type Documentation	23
5.1.4 Function Documentation	24
<b>6 Example Documentation</b>	<b>107</b>
6.1 hello_world.cpp	107
6.2 multiple_devices.cpp	108
6.3 single_device.cpp	109
Index	111



# 1 DHD - Haptic SDK Documentation

## 1.1 Introduction

This document provides an on-line description of the DHD calls and the related functionalities of the Force Dimension haptic devices. It presents a detailed syntax of each function and its arguments, and explains the programming concepts and features that will help the programmer get the best performance out of Force Dimension haptic devices.

### 1.1.1 Multi-platforms

The DHD is transparently multi-platform. Currently, implementations exist on

- *Microsoft Windows*
- *Linux*
- *Apple macOS*

The DHD is also available for the following Real-Time Operating Systems (RTOS):

- *Blackberry QNX*
- *Wind River VxWorks*
- *Microsoft Windows CE7*

The only requirement imposed by the multi-platform architecture of the DHD is the use of a preprocessor directive that matches your target system, as indicated in [dhdc.h](#) :

- **WIN32** must be defined for Microsoft Windows 32-bit compilation
- **WIN64** must be defined for Microsoft Windows 64-bit platforms
- **LINUX** must be defined for Linux platforms
- **MACOSX** must be defined for Apple MacOS X platforms
- **QNX** must be defined for QNX platforms
- **VXWORKS** must be defined for VxWorks platforms

### 1.1.2 High-level vs. Low-level SDK

The DHD is designed with two purposes in mind:

- to offer a simple and straightforward software library for programmers to interface their haptic device with their application with just a [few lines of code](#)
- to offer advanced control functionalities for experienced users who wish to write advanced control routines and adjust low-level parameters

The following sections describe the basic device features from a software perspective.

## 1.2 Features

### 1.2.1 Device Types

This version of the DHD can be used with the following devices:

- the second generation DELTA.X Haptic Devices
  - [DHD\\_DEVICE\\_DELTA3](#)
- the second generation OMEGA.X Haptic Devices
  - [DHD\\_DEVICE\\_OMEGA3](#)
  - [DHD\\_DEVICE\\_OMEGA33](#)
  - [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
  - [DHD\\_DEVICE\\_OMEGA331](#)
  - [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- the SIGMA.X Haptic Devices
  - [DHD\\_DEVICE\\_SIGMA331](#)
  - [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- the LAMBDA.X Haptic Devices
  - [DHD\\_DEVICE\\_LAMBDA331](#)
  - [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)
- the Force Dimension stand-alone USB 2.0 controller
  - [DHD\\_DEVICE\\_CONTROLLER](#)
  - [DHD\\_DEVICE\\_CONTROLLER\\_HR](#)
- the Novint FALCON haptic device
  - [DHD\\_DEVICE\\_FALCON](#)

Unknown devices that comply with the same protocol are referenced by [DHD\\_DEVICE\\_CUSTOM](#).

### 1.2.2 Axis Convention

Unless otherwise specified (e.g. for a specific device), the following convention is used when passing Cartesian data to a function in an array of the form:

```
double array[DHD_MAX_DOF]
```

- For positions and rotations:
  - Position data is stored in  
`array[0], array[1], array[2]`
  - Euler angles are stored in  
`array[3], array[4], array[5]`
  - Gripper opening is stored as the gripper opening distance in  
`array[6]`
- For velocities:
  - Linear (Cartesian) velocity is stored in  
`array[0], array[1], array[2]`
  - Angular (Cartesian) velocity are stored in  
`array[3], array[4], array[5]`

- Gripper opening distance velocity is stored in  
`array[6]`
- For forces and torques:
  - Cartesian force is stored as a vector  
`array[0], array[1], array[2]`
  - Cartesian torque is stored as a vector  
`array[3], array[4], array[5]`
  - Gripper force is stored in  
`array[6]`

### 1.2.3 Device Modes

When a device is active (powered ON), it is in one of the four states or 'modes' described below. For additional information, please refer to the user manuals.

- **RESET mode**  
In this mode, the user is expected to put the device end-effector at its rest position. This is how the device performs its calibration. A calibration can be explicitly required by calling `dhdReset()`.
- **IDLE mode**  
In this mode, the position of the end-effector can be read, but no current is applied to the device motors. This is a safe way to debug an application, or to use the device as a pointer. The device can be forced into IDLE mode by disabling the brakes via `dhdSetBrakes()`.
- **FORCE mode**  
In this mode, the device motors are enabled so that forces and optionally torques (for 6DOF devices) can be applied.
- **BRAKE mode**  
In this mode, electromagnetic braking is applied on the motors. As a result, there is added viscosity that prevents the end-effector from moving rapidly. This mode is entered when forces are disabled, or if a [safety features](#) triggers it.

### 1.2.4 Device Status

Force Dimension haptic devices status can be retrieved via the `dhdGetStatus()` function. The function returns a status vector containing the following fields:

- `DHD_STATUS_POWER`
- `DHD_STATUS_CONNECTED`
- `DHD_STATUS_STARTED`
- `DHD_STATUS_RESET`
- `DHD_STATUS_IDLE`
- `DHD_STATUS_FORCE`
- `DHD_STATUS_BRAKE`
- `DHD_STATUS_TORQUE`
- `DHD_STATUS_WRIST_DETECTED`
- `DHD_STATUS_ERROR`
- `DHD_STATUS_GRAVITY`
- `DHD_STATUS_TIMEGUARD`
- `DHD_STATUS_WRIST_INIT`
- `DHD_STATUS_REDUNDANCY`
- `DHD_STATUS_FORCE_OFF_CAUSE`
- `DHD_STATUS_LOCKS`
- `DHD_STATUS_AXIS_CHECKED`

### 1.2.5 Support for Multiple Devices

The DHD supports as many haptic devices connected to the same computer as the underlying operating system can accommodate. Once a device is opened, it receives an ID that uniquely identifies it within the SDK. The device that receives the commands from the SDK can be identified and selected at any time by calling [dhdGetDeviceID\(\)](#) and [dhdSetDevice\(\)](#). Also, every device specific function of the SDK can take as a last argument the device ID. If no last argument is given, or if that last argument is -1 (the default), the [default device](#) is used.

- [single device programming example](#)
- [multiple devices programming example](#)

### 1.2.6 Velocity Estimator

The SDK provides internal mechanisms that estimate the velocity of the device in the joint and cartesian coordinate systems. The default velocity estimator configuration should be suitable for most use cases, but can be reconfigured by calling:

- [dhdConfigLinearVelocity\(\)](#)
- [dhdConfigAngularVelocity\(\)](#)
- [dhdConfigGripperVelocity\(\)](#)

The estimated velocity can be retrieved by calling:

- [dhdGetLinearVelocity\(\)](#)
- [dhdGetAngularVelocityRad\(\)](#)
- [dhdGetAngularVelocityDeg\(\)](#)
- [dhdGetGripperLinearVelocity\(\)](#)
- [dhdGetGripperAngularVelocityRad\(\)](#)
- [dhdGetGripperAngularVelocityDeg\(\)](#)

Note that in this release, velocity is computed using [DHD\\_VELOCITY\\_WINDOWING](#) mode.

### 1.2.7 TimeGuard

The DHD features a throttling mechanism to provide a controllable communication refresh rate while preserving resources on non real-time OS. This mechanism prevents the OS from querying the device for its position at a rate higher than an adjustable threshold. In order to do so, TimeGuard prevents the application from requesting new position data if recent data from an earlier communication event is still recent enough. This mechanism can remove communication overhead without affecting performance if set properly, but can also significantly affect performance if set to the wrong value. It is recommended to leave the TimeGuard feature to its default setting unless a specific software architecture requires it. SDK calls that trigger the TimeGuard feature will return [DHD\\_TIMEGUARD](#) if communication with the device was not necessary, 0 otherwise (or see [error management](#) for possible return values). See [dhdSetTimeGuard\(\)](#) to adjust this feature.

### 1.2.8 Thread-Safe Operation

Every module of the SDK is thread-safe. Programmers need not add their own synchronizing mechanism to control access to the device or its geometric model.



### 1.2.9 Multi-threading

Multi-threading operation is fully supported by the SDK (see [Thread-Safe Operation](#) above). The SDK provides a simple convenience function ([dhdStartThread\(\)](#)) to start threads with a portable, operating system independent syntax. For more complex thread management, Force Dimension recommends using the native thread libraries of each operating system.

The [dhdStartThread\(\)](#) function allows to start any C-like function in a separate thread, with an optional argument and a given priority level. The priority level is defined in a portable, operating system independent way as:

- [DHD\\_THREAD\\_PRIORITY\\_DEFAULT](#)
- [DHD\\_THREAD\\_PRIORITY\\_HIGH](#)
- [DHD\\_THREAD\\_PRIORITY\\_LOW](#)

### 1.2.10 Error Management

The DHD uses a thread-safe global accessible via [dhdErrorGetLast\(\)](#), to store the last error that occurred in each running thread. Most functions and methods will return either 0 or a valid, positive value on success, and -1 (or NULL) in case of failure. On failure, programmers can check the value of [dhdErrorGetLast\(\)](#) against the [error values](#).

To help identify the error, the [dhdErrorGetStr\(\)](#) functions return a short descriptive string. Similarly, [dhdErrorGetLastStr\(\)](#) returns a string describing the last error that occurred within the calling thread.

The following functions can be used to retrieve error codes and their descriptions:

- [dhdErrorGetLast\(\)](#)
- [dhdErrorGetLastStr\(\)](#)
- [dhdErrorGetStr\(\)](#)

### 1.2.11 Safety Feature

As Force Dimension haptic devices can generate a significant amount of force, it could accelerate to a point that may damage the system, or surprise unaware users. To prevent such situations, the controller factory settings offer a safety feature that forces the device into BRAKE mode if the velocity becomes greater than a given [threshold](#). While it is possible to modify this value using advanced features from this SDK, it is recommended to keep this threshold as low as the application requires.

### 1.2.12 Units

Here is an overview of the units used in the SDK, unless otherwise specified:

- *length* : meter [m]
- *angles* : radian [rad] or [deg] (specified)
- *forces* : newton [N]
- *torques* : newton.meter [Nm]
- *time* : microsecond [us]

### 1.2.13 Standard SDK

The following functions can be called at any time.

- `dhdEnableSimulator()`
- `dhdGetDeviceCount()`
- `dhdGetAvailableCount()`
- `dhdSetDevice()`
- `dhdGetDeviceID()`
- `dhdGetSerialNumber()`
- `dhdOpen()`
- `dhdOpenType()`
- `dhdOpenSerial()`
- `dhdOpenID()`
- `dhdClose()`
- `dhdCheckControllerMemory()`
- `dhdStop()`
- `dhdGetComMode()`
- `dhdEnableForce()`
- `dhdEnableGripperForce()`
- `dhdGetSystemType()`
- `dhdGetSystemName()`
- `dhdGetSystemRev()`
- `dhdGetVersion()`
- `dhdGetVersionStr()`
- `dhdGetSDKVersion()`
- `dhdGetSDKVersionStr()`
- `dhdGetComponentVersionStr()`
- `dhdGetStatus()`
- `dhdGetDeviceAngleRad()`
- `dhdGetDeviceAngleDeg()`
- `dhdGetEffectorMass()`
- `dhdGetSystemCounter()`
- `dhdGetButton()`
- `dhdGetButtonMask()`
- `dhdSetOutput()`
- `dhdIsLeftHanded()`
- `dhdHasBase()`

- `dhdHasWrist()`
- `dhdHasActiveWrist()`
- `dhdHasGripper()`
- `dhdHasActiveGripper()`
- `dhdReset()`
- `dhdResetWrist()`
- `dhdWaitForReset()`
- `dhdSetStandardGravity()`
- `dhdSetGravityCompensation()`
- `dhdSetBrakes()`
- `dhdSetDeviceAngleRad()`
- `dhdSetDeviceAngleDeg()`
- `dhdSetEffectorMass()`
- `dhdGetPosition()`
- `dhdGetForce()`
- `dhdSetForce()`
- `dhdGetOrientationRad()`
- `dhdGetOrientationDeg()`
- `dhdGetPositionAndOrientationRad()`
- `dhdGetPositionAndOrientationDeg()`
- `dhdGetPositionAndOrientationFrame()`
- `dhdGetForceAndTorque()`
- `dhdSetForceAndTorque()`
- `dhdGetOrientationFrame()`
- `dhdGetGripperAngleDeg()`
- `dhdGetGripperAngleRad()`
- `dhdGetGripperGap()`
- `dhdGetGripperThumbPos()`
- `dhdGetGripperFingerPos()`
- `dhdGetComFreq()`
- `dhdSetForceAndGripperForce()`
- `dhdSetForceAndTorqueAndGripperForce()`
- `dhdGetForceAndTorqueAndGripperForce()`
- `dhdConfigLinearVelocity()`
- `dhdGetLinearVelocity()`
- `dhdConfigAngularVelocity()`
- `dhdGetAngularVelocityRad()`

- `dhdGetAngularVelocityDeg()`
- `dhdConfigGripperVelocity()`
- `dhdGetGripperLinearVelocity()`
- `dhdGetGripperAngularVelocityRad()`
- `dhdGetGripperAngularVelocityDeg()`
- `dhdEmulateButton()`
- `dhdGetBaseAngleXRad()`
- `dhdGetBaseAngleXDeg()`
- `dhdSetBaseAngleXRad()`
- `dhdSetBaseAngleXDeg()`
- `dhdGetBaseAngleZRad()`
- `dhdGetBaseAngleZDeg()`
- `dhdSetBaseAngleZRad()`
- `dhdSetBaseAngleZDeg()`
- `dhdSetVibration()`
- `dhdSetMaxForce()`
- `dhdSetMaxTorque()`
- `dhdSetMaxGripperForce()`
- `dhdGetMaxForce()`
- `dhdGetMaxTorque()`
- `dhdGetMaxGripperForce()`

#### 1.2.14 Expert SDK

This SDK offers high-level functions and methods that make it very easy to interface a Force Dimension haptic Device with any application. However, in some cases, users might want to get direct access to lower-level functionalities of the device (such as encoder readings and direct motor command). The SDK allows advanced users to access these routines by enabling the **expert** mode. Please note that the **expert** mode is for experienced programmers who have a thorough understanding of their haptic interface. Force Dimension cannot be held responsible for any damage resulting from use of the **expert** mode. The following functions are part of the expert SDK and require a deep understanding of control theory, as well as of the device design itself.

**USE AT YOUR OWN RISK !**

- `dhdEnableExpertMode()`
- `dhdDisableExpertMode()`
- `dhdPreset()`
- `dhdCalibrateWrist()`
- `dhdSetTimeGuard()`
- `dhdSetVelocityThreshold()`
- `dhdGetVelocityThreshold()`
- `dhdUpdateEncoders()`

- [dhdGetDeltaEncoders\(\)](#)
- [dhdGetWristEncoders\(\)](#)
- [dhdGetGripperEncoder\(\)](#)
- [dhdGetEncoder\(\)](#)
- [dhdSetMotor\(\)](#)
- [dhdSetDeltaMotor\(\)](#)
- [dhdSetWristMotor\(\)](#)
- [dhdSetGripperMotor\(\)](#)
- [dhdDeltaEncoderToPosition\(\)](#)
- [dhdDeltaPositionToEncoder\(\)](#)
- [dhdDeltaMotorToForce\(\)](#)
- [dhdDeltaForceToMotor\(\)](#)
- [dhdWristEncoderToOrientation\(\)](#)
- [dhdWristOrientationToEncoder\(\)](#)
- [dhdWristMotorToTorque\(\)](#)
- [dhdWristTorqueToMotor\(\)](#)
- [dhdGripperEncoderToAngleRad\(\)](#)
- [dhdGripperEncoderToGap\(\)](#)
- [dhdGripperAngleRadToEncoder\(\)](#)
- [dhdGripperGapToEncoder\(\)](#)
- [dhdGripperMotorToForce\(\)](#)
- [dhdGripperForceToMotor\(\)](#)
- [dhdSetMot\(\)](#)
- [dhdPreloadMot\(\)](#)
- [dhdGetEnc\(\)](#)
- [dhdGetEncRange\(\)](#)
- [dhdSetBrk\(\)](#)
- [dhdGetDeltaJointAngles\(\)](#)
- [dhdGetDeltaJacobian\(\)](#)
- [dhdDeltaJointAnglesToJacobian\(\)](#)
- [dhdDeltaJointTorquesExtrema\(\)](#)
- [dhdDeltaGravityJointTorques\(\)](#)
- [dhdSetDeltaJointTorques\(\)](#)
- [dhdDeltaEncodersToJointAngles\(\)](#)
- [dhdDeltaJointAnglesToEncoders\(\)](#)
- [dhdGetWristJointAngles\(\)](#)
- [dhdGetWristJacobian\(\)](#)

- [dhdWristJointAnglesToJacobian\(\)](#)
- [dhdWristJointTorquesExtrema\(\)](#)
- [dhdWristGravityJointTorques\(\)](#)
- [dhdSetWristJointTorques\(\)](#)
- [dhdSetForceAndWristJointTorques\(\)](#)
- [dhdSetForceAndWristJointTorquesAndGripperForce\(\)](#)
- [dhdWristEncodersToJointAngles\(\)](#)
- [dhdWristJointAnglesToEncoders\(\)](#)
- [dhdGetJointAngles\(\)](#)
- [dhdGetJointAngleRange\(\)](#)
- [dhdGetJointVelocities\(\)](#)
- [dhdGetEncVelocities\(\)](#)
- [dhdJointAnglesToInertiaMatrix\(\)](#)
- [dhdSetComMode\(\)](#)
- [dhdSetComModePriority\(\)](#)
- [dhdSetWatchdog\(\)](#)
- [dhdGetWatchdog\(\)](#)

### 1.2.15 Controller SDK

The following functions only apply to the [DHD\\_DEVICE\\_CONTROLLER](#) and [DHD\\_DEVICE\\_CONTROLLER\\_HR](#) devices.

- [dhdControllerSetDevice\(\)](#)
- [dhdReadConfigFromFile\(\)](#)

### 1.2.16 OS-independent SDK

- [dhdKbHit\(\)](#)
- [dhdKbGet\(\)](#)
- [dhdGetTime\(\)](#)
- [dhdSleep\(\)](#)
- [dhdStartThread\(\)](#)

## 1.3 Technical Support

Please contact your distributor for any technical support inquiry.

## 2 GLOSSARY

This page describes some of the technical expressions commonly used in the documentation. The expressions listed below regroup SDK features and technical definitions relevant to the field of haptics and control theory.

### Initialization

Initialization is necessary to obtain accurate, reproducible localization of the end-effector within the workspace of the device. Force Dimension haptic devices are designed in such a way that there can be no drift of the calibration over time, so the procedure only needs to be performed once when the device is powered on. The calibration procedure consists in placing the calibration pole in the dedicated calibration pit. The device detects when the calibration position is reached and the status LED stops blinking.

### Controller

The electronic controller is responsible for the real-time behavior of the device. It connects to the host computer and provides the low-level safety features such as [velocity thresholding](#) and communication timeouts.

### Default Device

In a [multiple devices](#) utilization, the SDK keeps an internal ID of one of the devices. All the SDK calls that do not explicitly mention a device ID are directed to the default device. The default device can be determined by calling [dhdGetDeviceID\(\)](#). The default device can be changed by calling [dhdSetDevice\(\)](#). Calls to [dhdOpen\(\)](#) change the default device ID to the last successfully opened device.

### Electromagnetic Brakes

In [BRAKES mode](#), the device motor circuits are shortcut to produce electromagnetic viscosity. The viscosity is sufficient to prevent the device from falling too hard onto if forces are disabled abruptly, either by pressing the force button or by action of a [safety feature](#).

### Gravity Compensation

To prevent user fatigue and to increase accuracy during manipulation, Force Dimension haptic devices features gravity compensation. When gravity compensation is enabled, the weights of the arms and of the end-effector are taken into account and a vertical force is dynamically applied to the end-effector on top of the user command. Please note that gravity compensation is computed on the host computer, and therefore only gets applied whenever a force command is sent to the device by the application. By default, gravity compensation is enabled and [dhdSetForce\(\)](#) compensates for the device weight. Gravity compensation can be disabled by calling [dhdSetGravityCompensation\(\)](#).

### Single Device Calls

When used with a single Force Dimension haptic device, programmers should use the single device version of the functions. Single device calls use the null [default device](#) ID, unlike the [multiple devices](#) SDK calls, which explicitly take the device ID as a last argument.

### Velocity Threshold

Every Force Dimension haptic device features a [safety feature](#) that prevents the device from accelerating without control. If the control unit detects that the velocity of the end-effector is higher than the programmed security limit, the forces are automatically disabled and the [device brakes](#) are engaged to prevent a possibly dangerous acceleration from the device. This velocity threshold can be adjusted or removed by calling [dhdSetVelocityThreshold\(\)](#).

### Watchdog Threshold

Force Dimension haptic devices with firmware version greater or equal to 3.0 features a [safety feature](#) that disables forces on the device if no communication is received by the controller for a given amount of time. If the control unit does not receive an expected input, the forces are automatically disabled and the [device brakes](#) are engaged to prevent potentially dangerous device behavior. This time duration of the watchdog feature can be adjusted or removed by calling [dhdSetWatchdog\(\)](#).

### Wrist Calibration

For 6 DOF Force Dimension devices, the [controller](#) performs a calibration procedure at power-up. This procedure is fully automated and does not require any user intervention during the few seconds it lasts. The calibration can be repeated without power-cycling the device by calling [dhdCalibrateWrist\(\)](#).

### COM operating Mode

USB operations can be executed in two different modes: [DHD\\_COM\\_MODE\\_SYNC](#) and [DHD\\_COM\\_MODE\\_ASYNC](#). Other operation modes are reported for virtual devices ([DHD\\_COM\\_MODE\\_VIRTUAL](#)) and devices that are connected over the network ([DHD\\_COM\\_MODE\\_NETWORK](#)). Please check the documentation of each mode for details.

## 3 Deprecated List

### Member `dhdCalibrateWrist` (char ID=-1)

This function is deprecated and is kept for backward compatibility only.

### Member `dhdGetSystemCounter` ()

This function is deprecated and is kept for backward compatibility only.

### Member `dhdResetWrist` (char ID=-1)

This function is deprecated and is kept for backward compatibility only.

## 4 File Index

### 4.1 File List

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

`dhdc.h`

DHD header file

12

## 5 File Documentation

### 5.1 `dhdc.h` File Reference

DHD header file.

#### Macros

- #define `DHD_DEVICE_NONE` 0
- #define `DHD_DEVICE_DELTA3` 63
- #define `DHD_DEVICE_OMEGA3` 33
- #define `DHD_DEVICE_OMEGA33` 34
- #define `DHD_DEVICE_OMEGA33_LEFT` 36
- #define `DHD_DEVICE_OMEGA331` 35
- #define `DHD_DEVICE_OMEGA331_LEFT` 37
- #define `DHD_DEVICE_FALCON` 60
- #define `DHD_DEVICE_CONTROLLER` 81
- #define `DHD_DEVICE_CONTROLLER_HR` 82
- #define `DHD_DEVICE_CUSTOM` 91
- #define `DHD_DEVICE_SIGMA331` 104
- #define `DHD_DEVICE_SIGMA331_LEFT` 105
- #define `DHD_DEVICE_LAMBDA331` 108
- #define `DHD_DEVICE_LAMBDA331_LEFT` 109
- #define `DHD_ON` 1
- #define `DHD_OFF` 0
- #define `DHD_UNDEFINED` -1
- #define `DHD_MAX_DOF` 8
- #define `DHD_DELTA_MOTOR_0` 0
- #define `DHD_DELTA_MOTOR_1` 1
- #define `DHD_DELTA_MOTOR_2` 2
- #define `DHD_DELTA_ENC_0` 0



- #define DHD\_DELTA\_ENC\_1 1
- #define DHD\_DELTA\_ENC\_2 2
- #define DHD\_WRIST\_MOTOR\_0 3
- #define DHD\_WRIST\_MOTOR\_1 4
- #define DHD\_WRIST\_MOTOR\_2 5
- #define DHD\_WRIST\_ENC\_0 3
- #define DHD\_WRIST\_ENC\_1 4
- #define DHD\_WRIST\_ENC\_2 5
- #define DHD\_TIMEGUARD 1
- #define DHD\_MOTOR\_SATURATED 2
- #define DHD\_MAX\_STATUS 17
- #define DHD\_STATUS\_POWER 0
- #define DHD\_STATUS\_CONNECTED 1
- #define DHD\_STATUS\_STARTED 2
- #define DHD\_STATUS\_RESET 3
- #define DHD\_STATUS\_IDLE 4
- #define DHD\_STATUS\_FORCE 5
- #define DHD\_STATUS\_BRAKE 6
- #define DHD\_STATUS\_TORQUE 7
- #define DHD\_STATUS\_WRIST\_DETECTED 8
- #define DHD\_STATUS\_ERROR 9
- #define DHD\_STATUS\_GRAVITY 10
- #define DHD\_STATUS\_TIMEGUARD 11
- #define DHD\_STATUS\_WRIST\_INIT 12
- #define DHD\_STATUS\_REDUNDANCY 13
- #define DHD\_STATUS\_FORCE\_OFF\_CAUSE 14
- #define DHD\_STATUS\_LOCKS 15
- #define DHD\_STATUS\_AXIS\_CHECKED 16
- #define DHD\_MAX\_BUTTONS 16
- #define DHD\_VELOCITY\_WINDOWING 0
- #define DHD\_VELOCITY\_WINDOW 20
- #define DHD\_COM\_MODE\_SYNC 0
- #define DHD\_COM\_MODE\_ASYNC 1
- #define DHD\_COM\_MODE\_VIRTUAL 3
- #define DHD\_COM\_MODE\_NETWORK 4
- #define DHD\_THREAD\_PRIORITY\_DEFAULT 0
- #define DHD\_THREAD\_PRIORITY\_HIGH 1
- #define DHD\_THREAD\_PRIORITY\_LOW 2

## Enumerations

- enum dhd\_errors {  
DHD\_NO\_ERROR, DHD\_ERROR, DHD\_ERROR\_COM, DHD\_ERROR\_DHC\_BUSY,  
DHD\_ERROR\_NO\_DRIVER\_FOUND, DHD\_ERROR\_NO\_DEVICE\_FOUND, DHD\_ERROR\_NOT\_AVAILABLE,  
DHD\_ERROR\_TIMEOUT,  
DHD\_ERROR\_GEOMETRY, DHD\_ERROR\_EXPERT\_MODE\_DISABLED, DHD\_ERROR\_NOT\_IMPLEMENTED,  
DHD\_ERROR\_OUT\_OF\_MEMORY,  
DHD\_ERROR\_DEVICE\_NOT\_READY, DHD\_ERROR\_FILE\_NOT\_FOUND, DHD\_ERROR\_CONFIGURATION,  
DHD\_ERROR\_INVALID\_INDEX,  
DHD\_ERROR\_DEPRECATED, DHD\_ERROR\_NULL\_ARGUMENT, DHD\_ERROR\_REDUNDANT\_FAIL, DHD\_ERROR\_NOT\_EN,  
DHD\_ERROR\_DEVICE\_IN\_USE, DHD\_ERROR\_INVALID, DHD\_ERROR\_NO\_REGULATION }

## Functions

- `int __SDK dhdErrorGetLast ()`
- `const char *__SDK dhdErrorGetLastStr ()`
- `const char *__SDK dhdErrorGetStr (int error)`
- `void __SDK dhdEnableSimulator (bool on)`
- `int __SDK dhdGetDeviceCount ()`
- `int __SDK dhdGetAvailableCount ()`
- `int __SDK dhdSetDevice (char ID)`
- `int __SDK dhdGetDeviceID ()`
- `int __SDK dhdGetSerialNumber (ushort *sn, char ID=-1)`
- `int __SDK dhdOpen ()`
- `int __SDK dhdOpenType (int type)`
- `int __SDK dhdOpenSerial (int serial)`
- `int __SDK dhdOpenID (char ID)`
- `int __SDK dhdClose (char ID=-1)`
- `int __SDK dhdCheckControllerMemory (char ID=-1)`
- `int __SDK dhdStop (char ID=-1)`
- `int __SDK dhdGetComMode (char ID=-1)`
- `int __SDK dhdEnableForce (uchar val, char ID=-1)`
- `int __SDK dhdEnableGripperForce (uchar val, char ID=-1)`
- `int __SDK dhdGetSystemType (char ID=-1)`
- `int __SDK dhdGetSystemRev (char ID=-1)`
- `const char *__SDK dhdGetSystemName (char ID=-1)`
- `int __SDK dhdGetVersion (double *ver, char ID=-1)`
- `int __SDK dhdGetVersionStr (char *buffer, size_t size, char ID=-1)`
- `void __SDK dhdGetSDKVersion (int *major, int *minor, int *release, int *revision)`
- `const char *__SDK dhdGetSDKVersionStr ()`
- `int __SDK dhdGetComponentVersionStr (uint32_t component, char *buffer, size_t size, char ID=-1)`
- `int __SDK dhdGetStatus (int status[DHD_MAX_STATUS], char ID=-1)`
- `int __SDK dhdGetDeviceAngleRad (double *angle, char ID=-1)`
- `int __SDK dhdGetDeviceAngleDeg (double *angle, char ID=-1)`
- `int __SDK dhdGetEffectorMass (double *mass, char ID=-1)`
- `ulong __SDK dhdGetSystemCounter ()`
- `int __SDK dhdGetButton (int index, char ID=-1)`
- `uint __SDK dhdGetButtonMask (char ID=-1)`
- `int __SDK dhdSetOutput (uint output, char ID=-1)`
- `bool __SDK dhdIsLeftHanded (char ID=-1)`
- `bool __SDK dhdHasBase (char ID=-1)`
- `bool __SDK dhdHasWrist (char ID=-1)`
- `bool __SDK dhdHasActiveWrist (char ID=-1)`
- `bool __SDK dhdHasGripper (char ID=-1)`
- `bool __SDK dhdHasActiveGripper (char ID=-1)`
- `int __SDK dhdReset (char ID=-1)`
- `int __SDK dhdResetWrist (char ID=-1)`
- `int __SDK dhdWaitForReset (int timeout=0, char ID=-1)`
- `int __SDK dhdSetStandardGravity (double g, char ID=-1)`
- `int __SDK dhdSetGravityCompensation (int val=DHD_ON, char ID=-1)`
- `int __SDK dhdSetBrakes (int val=DHD_ON, char ID=-1)`
- `int __SDK dhdSetDeviceAngleRad (double angle, char ID=-1)`
- `int __SDK dhdSetDeviceAngleDeg (double angle, char ID=-1)`
- `int __SDK dhdSetEffectorMass (double mass, char ID=-1)`
- `int __SDK dhdGetPosition (double *px, double *py, double *pz, char ID=-1)`
- `int __SDK dhdGetForce (double *fx, double *fy, double *fz, char ID=-1)`
- `int __SDK dhdSetForce (double fx, double fy, double fz, char ID=-1)`
- `int __SDK dhdGetOrientationRad (double *oa, double *ob, double *og, char ID=-1)`
- `int __SDK dhdGetOrientationDeg (double *oa, double *ob, double *og, char ID=-1)`

- int \_\_SDK [dhdGetPositionAndOrientationRad](#) (double \*px, double \*py, double \*pz, double \*oa, double \*ob, double \*og, char ID=-1)
- int \_\_SDK [dhdGetPositionAndOrientationDeg](#) (double \*px, double \*py, double \*pz, double \*oa, double \*ob, double \*og, char ID=-1)
- int \_\_SDK [dhdGetPositionAndOrientationFrame](#) (double \*px, double \*py, double \*pz, double matrix[3][3], char ID=-1)
- int \_\_SDK [dhdGetForceAndTorque](#) (double \*fx, double \*fy, double \*fz, double \*tx, double \*ty, double \*tz, char ID=-1)
- int \_\_SDK [dhdSetForceAndTorque](#) (double fx, double fy, double fz, double tx, double ty, double tz, char ID=-1)
- int \_\_SDK [dhdGetOrientationFrame](#) (double matrix[3][3], char ID=-1)
- int \_\_SDK [dhdGetGripperAngleDeg](#) (double \*a, char ID=-1)
- int \_\_SDK [dhdGetGripperAngleRad](#) (double \*a, char ID=-1)
- int \_\_SDK [dhdGetGripperGap](#) (double \*g, char ID=-1)
- int \_\_SDK [dhdGetGripperThumbPos](#) (double \*px, double \*py, double \*pz, char ID=-1)
- int \_\_SDK [dhdGetGripperFingerPos](#) (double \*px, double \*py, double \*pz, char ID=-1)
- double \_\_SDK [dhdGetComFreq](#) (char ID=-1)
- int \_\_SDK [dhdSetForceAndGripperForce](#) (double fx, double fy, double fz, double fg, char ID=-1)
- int \_\_SDK [dhdSetForceAndTorqueAndGripperForce](#) (double fx, double fy, double fz, double tx, double ty, double tz, double fg, char ID=-1)
- int \_\_SDK [dhdGetForceAndTorqueAndGripperForce](#) (double \*fx, double \*fy, double \*fz, double \*tx, double \*ty, double \*tz, double \*f, char ID=-1)
- int \_\_SDK [dhdConfigLinearVelocity](#) (int ms=[DHD\\_VELOCITY\\_WINDOW](#), int mode=[DHD\\_VELOCITY\\_WINDOWING](#), char ID=-1)
- int \_\_SDK [dhdGetLinearVelocity](#) (double \*vx, double \*vy, double \*vz, char ID=-1)
- int \_\_SDK [dhdConfigAngularVelocity](#) (int ms=[DHD\\_VELOCITY\\_WINDOW](#), int mode=[DHD\\_VELOCITY\\_WINDOWING](#), char ID=-1)
- int \_\_SDK [dhdGetAngularVelocityRad](#) (double \*wx, double \*wy, double \*wz, char ID=-1)
- int \_\_SDK [dhdGetAngularVelocityDeg](#) (double \*wx, double \*wy, double \*wz, char ID=-1)
- int \_\_SDK [dhdConfigGripperVelocity](#) (int ms=[DHD\\_VELOCITY\\_WINDOW](#), int mode=[DHD\\_VELOCITY\\_WINDOWING](#), char ID=-1)
- int \_\_SDK [dhdGetGripperLinearVelocity](#) (double \*vg, char ID=-1)
- int \_\_SDK [dhdGetGripperAngularVelocityRad](#) (double \*wg, char ID=-1)
- int \_\_SDK [dhdGetGripperAngularVelocityDeg](#) (double \*wg, char ID=-1)
- int \_\_SDK [dhdEmulateButton](#) (uchar val, char ID=-1)
- int \_\_SDK [dhdGetBaseAngleXRad](#) (double \*angle, char ID=-1)
- int \_\_SDK [dhdGetBaseAngleXDeg](#) (double \*angle, char ID=-1)
- int \_\_SDK [dhdSetBaseAngleXRad](#) (double angle, char ID=-1)
- int \_\_SDK [dhdSetBaseAngleXDeg](#) (double angle, char ID=-1)
- int \_\_SDK [dhdGetBaseAngleZRad](#) (double \*angle, char ID=-1)
- int \_\_SDK [dhdGetBaseAngleZDeg](#) (double \*angle, char ID=-1)
- int \_\_SDK [dhdSetBaseAngleZRad](#) (double angle, char ID=-1)
- int \_\_SDK [dhdSetBaseAngleZDeg](#) (double angle, char ID=-1)
- int \_\_SDK [dhdSetVibration](#) (double freq, double amplitude, int type=0, char ID=-1)
- int \_\_SDK [dhdSetMaxForce](#) (double f, char ID=-1)
- int \_\_SDK [dhdSetMaxTorque](#) (double t, char ID=-1)
- int \_\_SDK [dhdSetMaxGripperForce](#) (double fg, char ID=-1)
- double \_\_SDK [dhdGetMaxForce](#) (char ID=-1)
- double \_\_SDK [dhdGetMaxTorque](#) (char ID=-1)
- double \_\_SDK [dhdGetMaxGripperForce](#) (char ID=-1)
- int \_\_SDK [dhdEnableExpertMode](#) ()
- int \_\_SDK [dhdDisableExpertMode](#) ()
- int \_\_SDK [dhdPreset](#) (int val[[DHD\\_MAX\\_DOF](#)], uchar mask, char ID=-1)
- int \_\_SDK [dhdCalibrateWrist](#) (char ID=-1)
- int \_\_SDK [dhdSetTimeGuard](#) (int us, char ID=-1)
- int \_\_SDK [dhdSetVelocityThreshold](#) (uint val, char ID=-1)
- int \_\_SDK [dhdGetVelocityThreshold](#) (uint \*val, char ID=-1)
- int \_\_SDK [dhdUpdateEncoders](#) (char ID=-1)
- int \_\_SDK [dhdGetDeltaEncoders](#) (int \*enc0, int \*enc1, int \*enc2, char ID=-1)

- int \_\_SDK dhdGetWristEncoders (int \*enc0, int \*enc1, int \*enc2, char ID=-1)
- int \_\_SDK dhdGetGripperEncoder (int \*enc, char ID=-1)
- int \_\_SDK dhdGetEncoder (int index, char ID=-1)
- int \_\_SDK dhdSetMotor (int index, ushort val, char ID=-1)
- int \_\_SDK dhdSetDeltaMotor (ushort mot0, ushort mot1, ushort mot2, char ID=-1)
- int \_\_SDK dhdSetWristMotor (ushort mot0, ushort mot1, ushort mot2, char ID=-1)
- int \_\_SDK dhdSetGripperMotor (ushort mot, char ID=-1)
- int \_\_SDK dhdDeltaEncoderToPosition (int enc0, int enc1, int enc2, double \*px, double \*py, double \*pz, char ID=-1)
- int \_\_SDK dhdDeltaPositionToEncoder (double px, double py, double pz, int \*enc0, int \*enc1, int \*enc2, char ID=-1)
- int \_\_SDK dhdDeltaMotorToForce (ushort mot0, ushort mot1, ushort mot2, int enc0, int enc1, int enc2, double \*fx, double \*fy, double \*fz, char ID=-1)
- int \_\_SDK dhdDeltaForceToMotor (double fx, double fy, double fz, int enc0, int enc1, int enc2, ushort \*mot0, ushort \*mot1, ushort \*mot2, char ID=-1)
- int \_\_SDK dhdWristEncoderToOrientation (int enc0, int enc1, int enc2, double \*oa, double \*ob, double \*og, char ID=-1)
- int \_\_SDK dhdWristOrientationToEncoder (double oa, double ob, double og, int \*enc0, int \*enc1, int \*enc2, char ID=-1)
- int \_\_SDK dhdWristMotorToTorque (ushort mot0, ushort mot1, ushort mot2, int enc0, int enc1, int enc2, double \*tx, double \*ty, double \*tz, char ID=-1)
- int \_\_SDK dhdWristTorqueToMotor (double ta, double tb, double tg, int enc0, int enc1, int enc2, ushort \*mot0, ushort \*mot1, ushort \*mot2, char ID=-1)
- int \_\_SDK dhdGripperEncoderToAngleRad (int enc, double \*a, char ID=-1)
- int \_\_SDK dhdGripperEncoderToGap (int enc, double \*g, char ID=-1)
- int \_\_SDK dhdGripperAngleRadToEncoder (double a, int \*enc, char ID=-1)
- int \_\_SDK dhdGripperGapToEncoder (double g, int \*enc, char ID=-1)
- int \_\_SDK dhdGripperMotorToForce (ushort mot, double \*f, int e[4], char ID=-1)
- int \_\_SDK dhdGripperForceToMotor (double f, ushort \*mot, int e[4], char ID=-1)
- int \_\_SDK dhdSetMot (ushort mot[DHD\_MAX\_DOF], uchar mask=0xff, char ID=-1)
- int \_\_SDK dhdPreloadMot (ushort mot[DHD\_MAX\_DOF], uchar mask=0xff, char ID=-1)
- int \_\_SDK dhdGetEnc (int enc[DHD\_MAX\_DOF], uchar mask=0xff, char ID=-1)
- int \_\_SDK dhdSetBrk (uchar mask=0xff, char ID=-1)
- int \_\_SDK dhdGetDeltaJointAngles (double \*j0, double \*j1, double \*j2, char ID=-1)
- int \_\_SDK dhdGetDeltaJacobian (double jcb[3][3], char ID=-1)
- int \_\_SDK dhdDeltaJointAnglesToJacobian (double j0, double j1, double j2, double jcb[3][3], char ID=-1)
- int \_\_SDK dhdDeltaJointTorquesExtrema (double j0, double j1, double j2, double minq[3], double maxq[3], char ID=-1)
- int \_\_SDK dhdDeltaGravityJointTorques (double j0, double j1, double j2, double \*q0, double \*q1, double \*q2, char ID=-1)
- int \_\_SDK dhdSetDeltaJointTorques (double t0, double t1, double t2, char ID=-1)
- int \_\_SDK dhdDeltaEncodersToJointAngles (int enc0, int enc1, int enc2, double \*j0, double \*j1, double \*j2, char ID=-1)
- int \_\_SDK dhdDeltaJointAnglesToEncoders (double j0, double j1, double j2, int \*enc0, int \*enc1, int \*enc2, char ID=-1)
- int \_\_SDK dhdGetWristJointAngles (double \*j0, double \*j1, double \*j2, char ID=-1)
- int \_\_SDK dhdGetWristJacobian (double jcb[3][3], char ID=-1)
- int \_\_SDK dhdWristJointAnglesToJacobian (double j0, double j1, double j2, double jcb[3][3], char ID=-1)
- int \_\_SDK dhdWristJointTorquesExtrema (double j0, double j1, double j2, double minq[3], double maxq[3], char ID=-1)
- int \_\_SDK dhdWristGravityJointTorques (double j0, double j1, double j2, double \*q0, double \*q1, double \*q2, char ID=-1)
- int \_\_SDK dhdSetWristJointTorques (double t0, double t1, double t2, char ID=-1)
- int \_\_SDK dhdSetForceAndWristJointTorques (double fx, double fy, double fz, double t0, double t1, double t2, char ID=-1)
- int \_\_SDK dhdSetForceAndWristJointTorquesAndGripperForce (double fx, double fy, double fz, double t0, double t1, double t2, double fg, char ID=-1)
- int \_\_SDK dhdWristEncodersToJointAngles (int enc0, int enc1, int enc2, double \*j0, double \*j1, double \*j2, char ID=-1)

- int \_\_SDK [dhdWristJointAnglesToEncoders](#) (double j0, double j1, double j2, int \*enc0, int \*enc1, int \*enc2, char ID=-1)
- int \_\_SDK [dhdGetJointAngles](#) (double j[DHD\_MAX\_DOF], char ID=-1)
- int \_\_SDK [dhdGetJointVelocities](#) (double v[DHD\_MAX\_DOF], char ID=-1)
- int \_\_SDK [dhdGetEncVelocities](#) (double v[DHD\_MAX\_DOF], char ID=-1)
- int \_\_SDK [dhdJointAnglesToInertiaMatrix](#) (double j[DHD\_MAX\_DOF], double inertia[6][6], char ID=-1)
- int \_\_SDK [dhdSetComMode](#) (int mode, char ID=-1)
- int \_\_SDK [dhdSetComModePriority](#) (int priority, char ID=-1)
- int \_\_SDK [dhdSetWatchdog](#) (unsigned char val, char ID=-1)
- int \_\_SDK [dhdGetWatchdog](#) (unsigned char \*val, char ID=-1)
- int \_\_SDK [dhdGetEncRange](#) (int encMin[DHD\_MAX\_DOF], int encMax[DHD\_MAX\_DOF], char ID=-1)
- int \_\_SDK [dhdGetJointAngleRange](#) (double jmin[DHD\_MAX\_DOF], double jmax[DHD\_MAX\_DOF], char ID=-1)
- int \_\_SDK [dhdControllerSetDevice](#) (int device, char ID=-1)
- int \_\_SDK [dhdReadConfigFromFile](#) (char \*filename, char ID=-1)
- bool \_\_SDK [dhdKbHit](#) ()
- char \_\_SDK [dhdKbGet](#) ()
- double \_\_SDK [dhdGetTime](#) ()
- void \_\_SDK [dhdSleep](#) (double sec)
- int \_\_SDK [dhdStartThread](#) (void \*func(void \*), void \*arg, int priority)

### 5.1.1 Detailed Description

DHD header file.

### 5.1.2 Macro Definition Documentation

#### 5.1.2.1 DHD\_COM\_MODE\_ASYNC `#define DHD_COM_MODE_ASYNC 1`

The asynchronous USB mode is the default. The asynchronous USB mode allows the operating system to parallelise the read and write operations on the USB port. This parallel operation improves refresh rate stability by reducing communication jitter. Other factors also influence USB performance, including the choice of operating system, machine load and program optimisation.

#### 5.1.2.2 DHD\_COM\_MODE\_NETWORK `#define DHD_COM_MODE_NETWORK 4`

This mode is reported when connected to a haptic device using the Force Dimension network connection mode.

#### 5.1.2.3 DHD\_COM\_MODE\_SYNC `#define DHD_COM_MODE_SYNC 0`

The synchronous USB mode performs USB read and write operations in sequence, allowing for a theoretical haptic refresh rate of 4 kHz. Please note that Other factors also influence USB performance, including the choice of operating system, machine load and program optimisation.

#### 5.1.2.4 DHD\_COM\_MODE\_VIRTUAL `#define DHD_COM_MODE_VIRTUAL 3`

This mode is reported when connected to a virtual device.

**5.1.2.5 DHD\_DELTA\_ENC\_0** `#define DHD_DELTA_ENC_0 0`

Array index for encoder 0 of the DELTA structure (expert mode only).

**5.1.2.6 DHD\_DELTA\_ENC\_1** `#define DHD_DELTA_ENC_1 1`

Array index for encoder 1 of the DELTA structure (expert mode only).

**5.1.2.7 DHD\_DELTA\_ENC\_2** `#define DHD_DELTA_ENC_2 2`

Array index for encoder 2 of the DELTA structure (expert mode only).

**5.1.2.8 DHD\_DELTA\_MOTOR\_0** `#define DHD_DELTA_MOTOR_0 0`

Array index for motor 0 of the DELTA structure (expert mode only).

**5.1.2.9 DHD\_DELTA\_MOTOR\_1** `#define DHD_DELTA_MOTOR_1 1`

Array index for motor 1 of the DELTA structure (expert mode only).

**5.1.2.10 DHD\_DELTA\_MOTOR\_2** `#define DHD_DELTA_MOTOR_2 2`

Array index for motor 2 of the DELTA structure (expert mode only).

**5.1.2.11 DHD\_DEVICE\_CONTROLLER** `#define DHD_DEVICE_CONTROLLER 81`

Device identifier for the Force Dimension stand-alone USB 2.0 controller device.

**5.1.2.12 DHD\_DEVICE\_CONTROLLER\_HR** `#define DHD_DEVICE_CONTROLLER_HR 82`

Device identifier for the Force Dimension stand-alone USB 2.0 controller device with high-resolution encoders (24bits).

**5.1.2.13 DHD\_DEVICE\_CUSTOM** `#define DHD_DEVICE_CUSTOM 91`

Device identifier for an unknown device compatible with the Force Dimension communication protocol.

**5.1.2.14 DHD\_DEVICE\_DELTA3** `#define DHD_DEVICE_DELTA3 63`

Device identifier for the Force Dimension DELTA.3 haptic device (USB version).

**5.1.2.15 DHD\_DEVICE\_FALCON** `#define DHD_DEVICE_FALCON 60`

Device identifier for the Novint FALCON haptic device.

**5.1.2.16 DHD\_DEVICE\_LAMBDA331** `#define DHD_DEVICE_LAMBDA331 108`

Device identifier for the right-handed version of the Force Dimension LAMBDA.7 haptic device.

**5.1.2.17 DHD\_DEVICE\_LAMBDA331\_LEFT** `#define DHD_DEVICE_LAMBDA331_LEFT 109`

Device identifier for the left-handed version of the Force Dimension LAMBDA.7 haptic device.

**5.1.2.18 DHD\_DEVICE\_NONE** `#define DHD_DEVICE_NONE 0`

Device identifier returned when no device is connected.

**5.1.2.19 DHD\_DEVICE\_OMEGA3** `#define DHD_DEVICE_OMEGA3 33`

Device identifier for the Force Dimension OMEGA.3 haptic device.

**5.1.2.20 DHD\_DEVICE\_OMEGA33** `#define DHD_DEVICE_OMEGA33 34`

Device identifier for the right-handed version of the Force Dimension OMEGA.6 haptic device.

**5.1.2.21 DHD\_DEVICE\_OMEGA331** `#define DHD_DEVICE_OMEGA331 35`

Device identifier for the right-handed version of the Force Dimension OMEGA.7 haptic device.

**5.1.2.22 DHD\_DEVICE\_OMEGA331\_LEFT** `#define DHD_DEVICE_OMEGA331_LEFT 37`

Device identifier for the left-handed version of the Force Dimension OMEGA.7 haptic device.

**5.1.2.23 DHD\_DEVICE\_OMEGA33\_LEFT** `#define DHD_DEVICE_OMEGA33_LEFT 36`

Device identifier for the left-handed version of the Force Dimension OMEGA.6 haptic device.

**5.1.2.24 DHD\_DEVICE\_SIGMA331** `#define DHD_DEVICE_SIGMA331 104`

Device identifier for the right-handed version of the Force Dimension SIGMA.7 haptic device.

**5.1.2.25 DHD\_DEVICE\_SIGMA331\_LEFT** `#define DHD_DEVICE_SIGMA331_LEFT 105`

Device identifier for the left-handed version of the Force Dimension SIGMA.7 haptic device.

**5.1.2.26 DHD\_MAX\_BUTTONS** `#define DHD_MAX_BUTTONS 16`

The maximum number of buttons the SDK can address on the Force Dimension haptic device.

**5.1.2.27 DHD\_MAX\_DOF** `#define DHD_MAX_DOF 8`

Maximum number of encoder channels that are available.

**5.1.2.28 DHD\_MAX\_STATUS** `#define DHD_MAX_STATUS 17`

The length of the status array. See [device status](#) for details.

**5.1.2.29 DHD\_MOTOR\_SATURATED** `#define DHD_MOTOR_SATURATED 2`

Return value used when at least one of the motors cannot deliver the requested torque. Motor groups are scaled in order to preserve force and torque direction over magnitude.

**5.1.2.30 DHD\_OFF** `#define DHD_OFF 0`

Applies to the [device status](#) values.

**5.1.2.31 DHD\_ON** `#define DHD_ON 1`

Applies to the [device status](#) values.

**5.1.2.32 DHD\_STATUS\_AXIS\_CHECKED** `#define DHD_STATUS_AXIS_CHECKED 16`

A bit-wise mask that indicates the validation status of each axis. The validation status of all device axes can be assessed by calling the `drdCheckInit()` function in the Force Dimension Robotic SDK (DRD). Each bit of the status value returned corresponds to a the validation status of the corresponding axis, e.g.:

```
uint8_t validationMask = static_cast<uint8_t>(status[DHD_STATUS_AXISCHECKED]);
for (int axis = 0; axis < DHD_MAX_DOF; i++)
{
    uint8_t axisMask = (0x01 << axis);
    if (!(validationMask & axisMask))
    {
        std::cout << "axis number " << axis << " not validated" << std::endl;
    }
}
```

**5.1.2.33 DHD\_STATUS\_BRAKE** `#define DHD_STATUS_BRAKE 6`

The index of the BRAKE flag in the status array. This flag indicates if the device is in BRAKE mode or not. See [device modes](#) for details.

**5.1.2.34 DHD\_STATUS\_CONNECTED** `#define DHD_STATUS_CONNECTED 1`

The index of the connection flag in the status array. This flag indicates if the device is connected or not.

**5.1.2.35 DHD\_STATUS\_ERROR** `#define DHD_STATUS_ERROR 9`

The index of the error flag in the status array. This flag indicates if the an error happened on the [device controller](#).

**5.1.2.36 DHD\_STATUS\_FORCE** `#define DHD_STATUS_FORCE 5`

The index of the FORCE flag in the status array. This flag indicates if the device is in FORCE mode or not. See [device modes](#) for details.



#### 5.1.2.37 DHD\_STATUS\_FORCE\_OFF\_CAUSE `#define DHD_STATUS_FORCE_OFF_CAUSE 14`

The event that caused forces to be disabled on the device (the last time forces were turned off). The event can be one of the following value:

- `DHD_FORCE_OFF_NONE` : nothing has caused forces to be turned off yet
- `DHD_FORCE_OFF_BUTTON` : the force button was pushed
- `DHD_FORCE_OFF_VELOCITY` : the [velocity threshold](#) was reached
- `DHD_FORCE_OFF_WATCHDOG` : the [communication watchdog](#) kicked in
- `DHD_FORCE_OFF_SOFTWARE` : the software requested forces to be turned off, e.g. [dhdEnableForce\(\)](#)
- `DHD_FORCE_OFF_USBDISCN` : the USB cable was disconnected
- `DHD_FORCE_OFF_DEADMAN` : the dead man switch was disconnected

Note that not all devices support all the force-disabling mechanisms listed above.

#### 5.1.2.38 DHD\_STATUS\_GRAVITY `#define DHD_STATUS_GRAVITY 10`

The index of the gravity flag in the status array. This flag indicates if the gravity compensation option is enabled or not.

#### 5.1.2.39 DHD\_STATUS\_IDLE `#define DHD_STATUS_IDLE 4`

The index of the IDLE flag in the status array. This flag indicates if the device is in IDLE mode or not. See [device modes](#) for details.

#### 5.1.2.40 DHD\_STATUS\_LOCKS `#define DHD_STATUS_LOCKS 15`

The status of the locks on supported devices. The value can be either [DHD\\_ON](#) if the locks are engaged, [DHD\\_OFF](#) if the locks are disengaged, or [DHD\\_UNDEFINED](#) if the status of the locks is unknown.

#### 5.1.2.41 DHD\_STATUS\_POWER `#define DHD_STATUS_POWER 0`

The index of the power flag in the status array. This flag indicates if the device is powered or not.

#### 5.1.2.42 DHD\_STATUS\_REDUNDANCY `#define DHD_STATUS_REDUNDANCY 13`

The status of the redundant encoder consistency check. For devices equipped with redundant encoders, a value of 1 indicates that the redundancy check is successful. A value of 0 is reported otherwise, or if the device does not feature redundant encoders.

#### 5.1.2.43 DHD\_STATUS\_RESET `#define DHD_STATUS_RESET 3`

The index of the RESET flag in the status array. This flag indicates if the device is in RESET mode or not. See [device modes](#) for details.

#### 5.1.2.44 DHD\_STATUS\_STARTED `#define DHD_STATUS_STARTED 2`

The index of the start flag in the status array. This flag indicates if the [device controller](#) is running.

**5.1.2.45 DHD\_STATUS\_TIMEGUARD** `#define DHD_STATUS_TIMEGUARD 11`

The index of the TimeGuard flag in the status array. This flag indicates if the TimeGuard feature is enabled or not. See [TimeGuard feature](#) for details.

**5.1.2.46 DHD\_STATUS\_TORQUE** `#define DHD_STATUS_TORQUE 7`

The index of the TORQUE flag in the status array. This flag indicates if the torques are active or not when the device is in FORCE mode. See [device modes](#) for details.

**5.1.2.47 DHD\_STATUS\_WRIST\_DETECTED** `#define DHD_STATUS_WRIST_DETECTED 8`

The index of the WRIST\_DETECTED flag in the status array. This flag indicates if the device has a wrist or not. See [device types](#) for details.

**5.1.2.48 DHD\_STATUS\_WRIST\_INIT** `#define DHD_STATUS_WRIST_INIT 12`

The index of the WRIST\_INIT flag in the status array. This flag indicates if the device wrist is initialized or not. See [device types](#) for details.

**5.1.2.49 DHD\_THREAD\_PRIORITY\_DEFAULT** `#define DHD_THREAD_PRIORITY_DEFAULT 0`

This constant can be used to tell the [dhdStartThread\(\)](#) function to use the default operating system priority level for the newly created thread.

**5.1.2.50 DHD\_THREAD\_PRIORITY\_HIGH** `#define DHD_THREAD_PRIORITY_HIGH 1`

This constant can be used to tell the [dhdStartThread\(\)](#) function to use a priority level higher than the default operating system priority for the newly created thread.

**5.1.2.51 DHD\_THREAD\_PRIORITY\_LOW** `#define DHD_THREAD_PRIORITY_LOW 2`

This constant can be used to tell the [dhdStartThread\(\)](#) function to use a priority level lower than the default operating system priority for the newly created thread.

**5.1.2.52 DHD\_TIMEGUARD** `#define DHD_TIMEGUARD 1`

Return value used when the [TimeGuard](#) feature prevented an unnecessary communication with the device.

**5.1.2.53 DHD\_UNDEFINED** `#define DHD_UNDEFINED -1`

Applies to the [device status](#) values.

**5.1.2.54 DHD\_VELOCITY\_WINDOW** `#define DHD_VELOCITY_WINDOW 20`

The default window size used by the [velocity estimator](#). The actual time interval (or "window") can be adjusted using [dhdConfigLinearVelocity](#), and should be modified to best suit the dynamic behavior of the device for a given application.

**5.1.2.55 DHD\_VELOCITY\_WINDOWING** `#define DHD_VELOCITY_WINDOWING 0`

The default [velocity estimator](#) mode. In this mode, the velocity is estimated by comparing the current position with the position a given time interval ago. This time interval (or "window") can be adjusted using [dhdConfigLinearVelocity](#), and should be modified to best suit the dynamic behavior of the device for a given application. The windowing estimator mode is the least resource intensive.

**5.1.2.56 DHD\_WRIST\_ENC\_0** `#define DHD_WRIST_ENC_0 3`

Array index for encoder 0 of the WRIST structure (expert mode only).

**5.1.2.57 DHD\_WRIST\_ENC\_1** `#define DHD_WRIST_ENC_1 4`

Array index for encoder 1 of the WRIST structure (expert mode only).

**5.1.2.58 DHD\_WRIST\_ENC\_2** `#define DHD_WRIST_ENC_2 5`

Array index for encoder 2 of the WRIST structure (expert mode only).

**5.1.2.59 DHD\_WRIST\_MOTOR\_0** `#define DHD_WRIST_MOTOR_0 3`

Array index for motor 0 of the WRIST structure (expert mode only).

**5.1.2.60 DHD\_WRIST\_MOTOR\_1** `#define DHD_WRIST_MOTOR_1 4`

Array index for motor 1 of the WRIST structure (expert mode only).

**5.1.2.61 DHD\_WRIST\_MOTOR\_2** `#define DHD_WRIST_MOTOR_2 5`

Array index for motor 2 of the WRIST structure (expert mode only).

**5.1.3 Enumeration Type Documentation**

---

## Enumerator

---

### 5.1.3.1 dhd\_errors enum dhd\_errors

See [error management](#) for more details on error management.

## Enumerator

DHD_NO_ERROR	No error occurred during processing. This is set as the default value in most of the SDK functions.
DHD_ERROR	Undocumented error (for use by custom devices).
DHD_ERROR_COM	There was a communication error between the current device <a href="#">controller</a> and the host computer.
DHD_ERROR_DHC_BUSY	The current device <a href="#">controller</a> is busy and cannot perform the requested task. This error code does not apply to controllers that connect using the USB protocol.
DHD_ERROR_NO_DRIVER_FOUND	A required device driver is not installed. Please refer to the user manual installation section.
DHD_ERROR_NO_DEVICE_FOUND	No supported device was detected.
DHD_ERROR_NOT_AVAILABLE	The command or feature is not available for the current device or in this SDK version.
DHD_ERROR_TIMEOUT	The operation timed out.
DHD_ERROR_GEOMETRY	One of the current device models (kinematic or force) reported an error.
DHD_ERROR_EXPERT_MODE_DISABLED	The command or feature is not available because <a href="#">dhdEnableExpertMode()</a> has not been called.
DHD_ERROR_NOT_IMPLEMENTED	The command or feature is not implemented for the current device.
DHD_ERROR_OUT_OF_MEMORY	Required memory could not be allocated.
DHD_ERROR_DEVICE_NOT_READY	The current device is not ready to process the requested command.
DHD_ERROR_FILE_NOT_FOUND	A required file is missing.
DHD_ERROR_CONFIGURATION	There was an error trying to read the calibration data from the current device <a href="#">controller</a> memory.
DHD_ERROR_INVALID_INDEX	An index passed to the function is outside the expected valid range.
DHD_ERROR_DEPRECATED	This feature, function or current device is marked deprecated.
DHD_ERROR_NULL_ARGUMENT	The function producing this error was passed an unexpected null pointer argument.
DHD_ERROR_REDUNDANT_FAIL	The redundant encoder integrity test failed. This error code only applies to devices with redundant encoders.
DHD_ERROR_NOT_ENABLED	A feature is not enabled for the current device.
DHD_ERROR_DEVICE_IN_USE	The current device is already in use by a function that requires exclusive access.
DHD_ERROR_INVALID	The function producing this error was passed an invalid or unexpected argument.
DHD_ERROR_NO_REGULATION	The robotic regulation thread is not running. This only applies to functions from the robotic SDK.

### 5.1.4 Function Documentation

**5.1.4.1 dhdCalibrateWrist()** `int __SDK dhdCalibrateWrist (`  
`char ID )`

This function triggers the [wrist calibration](#) routine in the [device controller](#).

#### Parameters

<i>ID</i>	[ <b>default=-1</b> ] device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

**Deprecated** This function is deprecated and is kept for backward compatibility only.

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.2 dhdCheckControllerMemory()** `int __SDK dhdCheckControllerMemory (`  
`char ID )`

This function evaluates the integrity of the device controller firmware and internal configuration on supported device types.

#### Parameters

<i>ID</i>	[ <b>default=-1</b> ] device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

#### Returns

[DHD\\_NO\\_ERROR](#) on success.  
[DHD\\_ERROR\\_CONFIGURATION](#) if the firmware or internal configuration health check failed.

**5.1.4.3 dhdClose()** `int __SDK dhdClose (`  
`char ID )`

This function closes the connection to a particular device.

#### Parameters

<i>ID</i>	[ <b>default=-1</b> ] device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

#### Examples

[hello\\_world.cpp](#), [multiple\\_devices.cpp](#), and [single\\_device.cpp](#).

**5.1.4.4 dhdConfigAngularVelocity()** `int __SDK dhdConfigAngularVelocity (`  
     `int ms,`  
     `int mode,`  
     `char ID )`

This function configures the internal velocity computation estimator. This only applies to the device wrist.

#### Parameters

<i>ms</i>	<b>[default=DHD_VELOCITY_WINDOW]</b> time interval used to compute velocity [ms]
<i>mode</i>	<b>[default=DHD_VELOCITY_WINDOWING]</b> device ID (see <a href="#">velocity estimator modes</a> section for details)
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### See also

[velocity estimator](#) for details  
[dhdGetAngularVelocityRad\(\)](#)  
[dhdGetAngularVelocityDeg\(\)](#)

#### Returns

0 on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.5 dhdConfigGripperVelocity()** `int __SDK dhdConfigGripperVelocity (`  
     `int ms,`  
     `int mode,`  
     `char ID )`

This function configures the internal velocity computation estimator. This only applies to the device gripper.

#### Parameters

<i>ms</i>	<b>[default=DHD_VELOCITY_WINDOW]</b> time interval used to compute velocity [ms]
<i>mode</i>	<b>[default=DHD_VELOCITY_WINDOWING]</b> device ID (see <a href="#">velocity estimator modes</a> section for details)
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### See also

[velocity estimator](#) for details  
[dhdGetGripperLinearVelocity\(\)](#)  
[dhdGetGripperAngularVelocityRad\(\)](#)  
[dhdGetGripperAngularVelocityDeg\(\)](#)

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.6 dhdcConfigLinearVelocity()** `int __SDK dhdcConfigLinearVelocity (`  
     `int ms,`  
     `int mode,`  
     `char ID )`

This function configures the internal velocity computation estimator. This only applies to the device base.

**Parameters**

<i>ms</i>	<b>[default=DHD_VELOCITY_WINDOW]</b> time interval used to compute velocity [ms]
<i>mode</i>	<b>[default=DHD_VELOCITY_WINDOWING]</b> device ID (see <a href="#">velocity estimator modes</a> section for details)
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**See also**

[dhdcGetLinearVelocity\(\)](#) and [velocity estimator](#) for details

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.7 dhdcControllerSetDevice()** `int __SDK dhdcControllerSetDevice (`  
     `int device,`  
     `char ID )`

If the connected device is a [controller](#), this function lets the programmer define the Force Dimension mechanical structure attached to it. Upon selecting a device model, the routine will attempt to read that particular device configuration from the controller. If this fails, a default configuration will be selected and stored in the controller.

**Parameters**

<i>device</i>	the <a href="#">device type</a> to use
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Note**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_CONTROLLER](#)
- [DHD\\_DEVICE\\_CONTROLLER\\_HR](#)

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.8 dhdDeltaEncodersToJointAngles()** `int __SDK dhdDeltaEncodersToJointAngles (`  
`int enc0,`  
`int enc1,`  
`int enc2,`  
`double * j0,`  
`double * j1,`  
`double * j2,`  
`char ID )`

This function computes and returns the delta joint angles for a given set of encoder readings.

**Parameters**

<i>enc0</i>	DELTA encoder reading on axis 0
<i>enc1</i>	DELTA encoder reading on axis 1
<i>enc2</i>	DELTA encoder reading on axis 2
<i>j0</i>	<b>[out]</b> joint angle for axis 0 in [rad]
<i>j1</i>	<b>[out]</b> joint angle for axis 1 in [rad]
<i>j2</i>	<b>[out]</b> joint angle for axis 2 in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Note**

**expert mode only - USE AT YOUR OWN RISKS**

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.9 dhdDeltaEncoderToPosition()** `int __SDK dhdDeltaEncoderToPosition (`  
`int enc0,`  
`int enc1,`  
`int enc2,`  
`double * px,`  
`double * py,`  
`double * pz,`  
`char ID )`

This function computes and returns the position of the end-effector for a given set of encoder readings.

**Parameters**

<i>enc0</i>	DELTA encoder reading on axis 0
<i>enc1</i>	DELTA encoder reading on axis 1



## Parameters

<i>enc2</i>	DELTA encoder reading on axis 2
<i>px</i>	<b>[out]</b> DELTA end-effector position on the X axis [m]
<i>py</i>	<b>[out]</b> DELTA end-effector position on the Y axis [m]
<i>pz</i>	<b>[out]</b> DELTA end-effector position on the Z axis [m]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

```

5.1.4.10 dhdcDeltaForceToMotor() int __SDK dhdcDeltaForceToMotor (
    double fx,
    double fy,
    double fz,
    int enc0,
    int enc1,
    int enc2,
    ushort * mot0,
    ushort * mot1,
    ushort * mot2,
    char ID )

```

This function computes and returns the motor commands necessary to obtain a given force on the end-effector at a given position (defined by encoder readings).

## Parameters

<i>fx</i>	force on the DELTA end-effector on the X axis [N]
<i>fy</i>	force on the DELTA end-effector on the Y axis [N]
<i>fz</i>	force on the DELTA end-effector on the Z axis [N]
<i>enc0</i>	DELTA encoder reading on axis 0
<i>enc1</i>	DELTA encoder reading on axis 1
<i>enc2</i>	DELTA encoder reading on axis 2
<i>mot0</i>	<b>[out]</b> motor command on DELTA axis 0
<i>mot1</i>	<b>[out]</b> motor command on DELTA axis 1
<i>mot2</i>	<b>[out]</b> motor command on DELTA axis 2
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 or [DHD\\_MOTOR\\_SATURATED](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.11 dhdDeltaGravityJointTorques()** `int __SDK dhdDeltaGravityJointTorques (`  
`double j0,`  
`double j1,`  
`double j2,`  
`double * q0,`  
`double * q1,`  
`double * q2,`  
`char ID )`

This function computes the DELTA joint torques required to compensate for gravity in a given DELTA joint angle configuration. Please refer to your device user manual for more information on your device coordinate system.

## Parameters

<i>j0</i>	joint angle for axis 0 in [rad]
<i>j1</i>	joint angle for axis 1 in [rad]
<i>j2</i>	joint angle for axis 2 in [rad]
<i>q0</i>	<b>out</b> gravity compensation joint torque on axis 0 in [Nm]
<i>q1</i>	<b>out</b> gravity compensation joint torque on axis 1 in [Nm]
<i>q2</i>	<b>out</b> gravity compensation joint torque on axis 2 in [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.12 dhdDeltaJointAnglesToEncoders()** `int __SDK dhdDeltaJointAnglesToEncoders (`  
`double j0,`  
`double j1,`  
`double j2,`  
`int * enc0,`  
`int * enc1,`  
`int * enc2,`  
`char ID )`

This function computes and returns the delta encoder readings for a given set of joint angles.

## Parameters

<i>j0</i>	joint angle for axis 0 in [rad]
-----------	---------------------------------

## Parameters

<i>j1</i>	joint angle for axis 1 in [rad]
<i>j2</i>	joint angle for axis 2 in [rad]
<i>enc0</i>	<b>[out]</b> DELTA encoder reading on axis 0
<i>enc1</i>	<b>[out]</b> DELTA encoder reading on axis 1
<i>enc2</i>	<b>[out]</b> DELTA encoder reading on axis 2
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

```
5.1.4.13 dhDeltaJointAnglesToJacobian() int __SDK dhDeltaJointAnglesToJacobian (
    double j0,
    double j1,
    double j2,
    double jcb[3][3],
    char ID )
```

This function retrieves the delta jacobian matrix based on a given joint configuration. Please refer to your device user manual for more information on your device coordinate system.

## Parameters

<i>j0</i>	joint angle for axis 0 in [rad]
<i>j1</i>	joint angle for axis 1 in [rad]
<i>j2</i>	joint angle for axis 2 in [rad]
<i>jcb</i>	<b>[out]</b> device jacobian
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.14 dhdDeltaJointTorquesExtrema()** `int __SDK dhdDeltaJointTorquesExtrema (`  
`double j0,`  
`double j1,`  
`double j2,`  
`double minq[3],`  
`double maxq[3],`  
`char ID )`

This function computes the range of applicable DELTA joint torques for a given DELTA joint angle configuration. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>j0</i>	joint angle for axis 0 in [rad]
<i>j1</i>	joint angle for axis 1 in [rad]
<i>j2</i>	joint angle for axis 2 in [rad]
<i>minq</i>	<b>outarray</b> of minimum applicable joint torque in [Nm]
<i>maxq</i>	<b>outarray</b> of maximum applicable joint torque in [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

#### Returns

0 on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.15 dhdDeltaMotorToForce()** `int __SDK dhdDeltaMotorToForce (`  
`ushort mot0,`  
`ushort mot1,`  
`ushort mot2,`  
`int enc0,`  
`int enc1,`  
`int enc2,`  
`double * fx,`  
`double * fy,`  
`double * fz,`  
`char ID )`

This function computes and returns the force applied to the end-effector for a given set of motor commands at a given position (defined by encoder readings).

#### Parameters

<i>mot0</i>	motor command on DELTA axis 0
<i>mot1</i>	motor command on DELTA axis 1
<i>mot2</i>	motor command on DELTA axis 2
<i>enc0</i>	DELTA encoder reading on axis 0
<i>enc1</i>	DELTA encoder reading on axis 1
<i>enc2</i>	DELTA encoder reading on axis 2
<i>fx</i>	<b>[out]</b> force on the DELTA end-effector on the X axis [N]
<i>fy</i>	<b>[out]</b> force on the DELTA end-effector on the Y axis [N]
<i>fz</i>	<b>[out]</b> force on the DELTA end-effector on the Z axis [N]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for detail.

**5.1.4.16 dhdDeltaPositionToEncoder()** `int __SDK dhdDeltaPositionToEncoder (`  
    `double px,`  
    `double py,`  
    `double pz,`  
    `int * enc0,`  
    `int * enc1,`  
    `int * enc2,`  
    `char ID )`

This function computes and returns the encoder values for a given end-effector position.

## Parameters

<i>px</i>	DELTA end-effector position on the X axis [m]
<i>py</i>	DELTA end-effector position on the Y axis [m]
<i>pz</i>	DELTA end-effector position on the Z axis [m]
<i>enc0</i>	<b>[out]</b> DELTA encoder reading on axis 0
<i>enc1</i>	<b>[out]</b> DELTA encoder reading on axis 1
<i>enc2</i>	<b>[out]</b> DELTA encoder reading on axis 2
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.17 dhdDisableExpertMode()** `int __SDK dhdDisableExpertMode ( )`

This function disables the [expert mode](#).

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.18 dhdEmulateButton()** `int __SDK dhdEmulateButton (`  
    `uchar val,`  
    `char ID )`

This function enables the button behavior emulation in devices that feature a gripper.

#### Parameters

<i>val</i>	DHD_ON to emulate button behavior, DHD_OFF to disable it
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

For omega.7 devices with firmware versions 2.x, forces need to be enabled for the button emulation to report the emulated button status.

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.19 dhdEnableExpertMode()** `int __SDK dhdEnableExpertMode ( )`

This function enables the [expert mode](#).

#### Note

**READ USER/PROGRAMMING MANUALS FIRST  
USE AT YOUR OWN RISKS**

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.20 dhdEnableForce()** `int __SDK dhdEnableForce (`  
    `uchar val,`  
    `char ID )`

This function enables the force mode in the [device controller](#).

## Parameters

<i>val</i>	DHD_ON to enable force, DHD_OFF to disable it
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.21 dhdEnableGripperForce()** `int __SDK dhdEnableGripperForce (`  
    `uchar val,`  
    `char ID )`

This function enables the gripper force mode in the [device controller](#). This function is only relevant to devices that have a gripper with a default closed or opened state. It does **not** apply to the sigma.x and omega.x range of devices, whose gripper does not have a default state. For those devices, the gripper force is enabled/disabled by [dhdEnableForce\(\)](#).

## Note

Forces must already have been enabled on the device (by calling [dhdEnableForce\(DHD\\_ON\)](#)) for [dhdEnableGripperForce\(\)](#) to have any effect.

## Parameters

<i>val</i>	DHD_ON to enable gripper force, DHD_OFF to disable it
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.22 dhdEnableSimulator()** `void __SDK dhdEnableSimulator (`  
    `bool on )`

This function enables the device simulator support. This enables network access on the loopback interface.

## Parameters

<i>on</i>	<b>true</b> to enable, <b>false</b> to disable
-----------	--

**5.1.4.23 dhdErrorGetLast()** `int __SDK dhdErrorGetLast ( )`

Returns the last error code encountered in the running thread. See [error management](#) for details.

**Returns**

The last error code encountered in the running thread.

**See also**

[dhdErrorGetStr \(\)](#)

**5.1.4.24 dhdErrorGetLastStr()** `const char* __SDK dhdErrorGetLastStr ( )`

Returns a brief string describing the last error encountered in the running thread. See [error management](#) for details.

**Returns**

A pointer to a character array.

**See also**

[dhdErrorGetStr \(\)](#)

**Examples**

[multiple\\_devices.cpp](#), and [single\\_device.cpp](#).

**5.1.4.25 dhdErrorGetStr()** `const char* __SDK dhdErrorGetStr (   
int error )`

Returns a brief string describing a given error code. See [error management](#) for details.

**Parameters**

<i>error</i>	error code
--------------	------------

**Returns**

A pointer to a character array.

**See also**

[dhdErrorGetLastStr \(\)](#)

**5.1.4.26 dhdGetAngularVelocityDeg()** `int __SDK dhdGetAngularVelocityDeg (   
double * wx,   
double * wy,`



```
double * wz,
char ID )
```

This function retrieves the estimated instantaneous angular velocity in [deg/s]. Velocity computation can be configured by calling [dhdConfigAngularVelocity\(\)](#). By default [DHD\\_VELOCITY\\_WINDOW](#) and [DHD\\_VELOCITY\\_WINDOWING](#) are used. See [velocity estimator](#) for details.

#### Parameters

<i>wx</i>	<b>[out]</b> angular velocity around the X axis [deg/s]
<i>wy</i>	<b>[out]</b> angular velocity around the Y axis [deg/s]
<i>wz</i>	<b>[out]</b> angular velocity around the Z axis [deg/s]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### See also

[dhdConfigAngularVelocity\(\)](#) and [velocity estimator](#) for details  
[dhdGetAngularVelocityRad\(\)](#)

#### Note

Please note that the velocity estimator requires at least 2 position updates during the time interval defined in [dhdConfigAngularVelocity\(\)](#) in order to be able to compute the estimate. Otherwise, e.g. if there are no calls to [dhdGetPosition\(\)](#) or [dhdGetAngularVelocityDeg\(\)](#) within the time interval window, [dhdGetAngularVelocityDeg\(\)](#) will return an error ([DHD\\_ERROR\\_TIMEOUT](#)).

#### Returns

0 on success, -1 on failure.  
 See [error management](#) for details.

#### 5.1.4.27 dhdGetAngularVelocityRad()

```
int __SDK dhdGetAngularVelocityRad (
double * wx,
double * wy,
double * wz,
char ID )
```

This function retrieves the estimated instantaneous angular velocity in [rad/s]. Velocity computation can be configured by calling [dhdConfigAngularVelocity\(\)](#). By default [DHD\\_VELOCITY\\_WINDOW](#) and [DHD\\_VELOCITY\\_WINDOWING](#) are used. See [velocity estimator](#) for details.

#### Parameters

<i>wx</i>	<b>[out]</b> angular velocity around the X axis [rad/s]
<i>wy</i>	<b>[out]</b> angular velocity around the Y axis [rad/s]
<i>wz</i>	<b>[out]</b> angular velocity around the Z axis [rad/s]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**See also**

[dhdConfigAngularVelocity\(\)](#) and [velocity estimator](#) for details  
[dhdGetAngularVelocityDeg\(\)](#)

**Note**

Please note that the velocity estimator requires at least 2 position updates during the time interval defined in [dhdConfigAngularVelocity\(\)](#) in order to be able to compute the estimate. Otherwise, e.g. if there are no calls to [dhdGetPosition\(\)](#) or [dhdGetAngularVelocityRad\(\)](#) within the time interval window, [dhdGetAngularVelocityRad\(\)](#) will return an error (`DHD_ERROR_TIMEOUT`).

**Returns**

0 on success, -1 on failure.  
See [error management](#) for details.

**5.1.4.28 dhdGetAvailableCount()** `int __SDK dhdGetAvailableCount ( )`

This function returns the number of available Force Dimension devices connected to the system. This encompasses all devices connected locally, but excludes devices already locked by other applications. Devices are given a unique identifier, as explained in the [multiple devices](#) section.

**See also**

[dhdGetDeviceCount\(\)](#)

**Returns**

the number of devices available on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.29 dhdGetBaseAngleXDeg()** `int __SDK dhdGetBaseAngleXDeg (`  
`double * angle,`  
`char ID )`

This function retrieves the device base plate angle around the X axis.

**Parameters**

<i>angle</i>	<b>[out]</b> a pointer to a valid <code>double</code> to receive the device angle in [deg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

See also

[dhdGetBaseAngleXRad\(\)](#)

**5.1.4.30 dhdGetBaseAngleXRad()** `int __SDK dhdGetBaseAngleXRad (`  
    `double * angle,`  
    `char ID )`

This function retrieves the device base plate angle around the X axis.

Parameters

<i>angle</i>	<b>[out]</b> a pointer to a valid <code>double</code> to receive the device angle in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

See also

[dhdGetBaseAngleXDeg\(\)](#)

**5.1.4.31 dhdGetBaseAngleZDeg()** `int __SDK dhdGetBaseAngleZDeg (`  
    `double * angle,`  
    `char ID )`

This function retrieves the device base plate angle around the vertical Z axis.

Parameters

<i>angle</i>	<b>[out]</b> a pointer to a valid <code>double</code> to receive the device angle in [deg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

See also

[dhdGetBaseAngleZRad\(\)](#)

**5.1.4.32 dhdGetBaseAngleZRad()** `int __SDK dhdGetBaseAngleZRad (`  
    `double * angle,`  
    `char ID )`

This function retrieves the device base plate angle around the vertical Z axis.

## Parameters

<i>angle</i>	<b>[out]</b> a pointer to a valid <code>double</code> to receive the device angle in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

## See also

[dhdGetBaseAngleZDeg\(\)](#)

**5.1.4.33 dhdGetButton()** `int __SDK dhdGetButton (`  
     `int index,`  
     `char ID )`

This function returns the status of the button located on the end-effector.

## Parameters

<i>index</i>	button index, 0 for the gripper button (up to <a href="#">DHD_MAX_BUTTONS</a> )
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

DHD\_ON if the button is pressed, DHD\_OFF otherwise, or -1 on error.  
See [error management](#) for details.

## Examples

[hello\\_world.cpp](#), [multiple\\_devices.cpp](#), and [single\\_device.cpp](#).

**5.1.4.34 dhdGetButtonMask()** `uint __SDK dhdGetButtonMask (`  
     `char ID )`

This function returns the 32-bit binary mask of the device buttons.

## Parameters

<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

## Returns

A 32-bit long bitmask. Each bit is set to 1 if the button is pressed, 0 otherwise.

**5.1.4.35 dhdGetComFreq()** `double __SDK dhdGetComFreq (`  
`char ID )`

This function returns the communication refresh rate between the computer and the device. Refresh rate computation is based on function calls that apply a force on the device (e.g. [dhdSetForce\(\)](#)).

#### Note

The refresh rate counters are reset every time the function is called. Therefore, it is recommended to call this function periodically (typically every second) in order to avoid discretization errors.

#### Parameters

<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

#### Returns

the refresh rate in [kHz], 0.0 otherwise.

**5.1.4.36 dhdGetComMode()** `int __SDK dhdGetComMode (`  
`char ID )`

This function retrieves the [COM operation mode](#) on compatible devices.

#### Parameters

<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

#### Returns

the current [COM operation mode](#) on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.37 dhdGetComponentVersionStr()** `int __SDK dhdGetComponentVersionStr (`  
`uint32_t component,`  
`char * buffer,`  
`size_t size,`  
`char ID )`

This function returns a string that describes an internal component version (if present).

#### Parameters

<i>component</i>	Component ID provided by Force Dimension (device-specific).
<i>buffer</i>	A user-allocated buffer to receive the component version string.
<i>size</i>	The size of the user-allocated version string buffer.
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Returns**

The version string on success, an empty string otherwise.

**5.1.4.38 dhdGetDeltaEncoders()** `int __SDK dhdGetDeltaEncoders (`  
`int * enc0,`  
`int * enc1,`  
`int * enc2,`  
`char ID )`

This function reads all encoders values of the DELTA structure.

**Parameters**

<i>enc0</i>	<b>[out]</b> DELTA axis 0 encoder reading
<i>enc1</i>	<b>[out]</b> DELTA axis 1 encoder reading
<i>enc2</i>	<b>[out]</b> DELTA axis 2 encoder reading
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Note**

**expert mode only - USE AT YOUR OWN RISKS**

**Returns**

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.39 dhdGetDeltaJacobian()** `int __SDK dhdGetDeltaJacobian (`  
`double jacobian[3][3],`  
`char ID )`

This function retrieves the jacobian matrix based on the current end-effector position. Please refer to your device user manual for more information on your device coordinate system.

**Parameters**

<i>jacobian</i>	<b>[out]</b> device jacobian
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Returns**

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.40 dhdGetDeltaJointAngles()** `int __SDK dhdGetDeltaJointAngles (`  
    `double * j0,`  
    `double * j1,`  
    `double * j2,`  
    `char ID )`

This function retrieves the joint angles in [rad] for the DELTA structure.

#### Parameters

<i>j0</i>	<b>[out]</b> joint angle for axis 0 in [rad]
<i>j1</i>	<b>[out]</b> joint angle for axis 1 in [rad]
<i>j2</i>	<b>[out]</b> joint angle for axis 2 in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

#### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.41 dhdGetDeviceAngleDeg()** `int __SDK dhdGetDeviceAngleDeg (`  
    `double * angle,`  
    `char ID )`

This function retrieves the device base plate angle around the Y axis.

#### Parameters

<i>angle</i>	<b>[out]</b> a pointer to a valid <code>double</code> to receive the device angle in [deg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

#### See also

[dhdGetDeviceAngleRad\(\)](#)

**5.1.4.42 dhdGetDeviceAngleRad()** `int __SDK dhdGetDeviceAngleRad (`  
    `double * angle,`  
    `char ID )`

This function retrieves the device base plate angle around the Y axis.



## Parameters

<i>angle</i>	<b>[out]</b> a pointer to a valid <code>double</code> to receive the device angle in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

## See also

[dhdGetDeviceAngleDeg\(\)](#)

#### 5.1.4.43 dhdGetDeviceCount() `int __SDK dhdGetDeviceCount ( )`

This function returns the number of compatible Force Dimension devices connected to the system. This encompasses all devices connected locally, including devices already locked by other applications. Devices are given a unique identifier, as explained in the [multiple devices](#) section.

## See also

[dhdGetAvailableCount\(\)](#)

## Returns

the number of devices connected on success, -1 otherwise.  
See [error management](#) for details.

## Examples

[multiple\\_devices.cpp](#), and [single\\_device.cpp](#).

#### 5.1.4.44 dhdGetDeviceID() `int __SDK dhdGetDeviceID ( )`

This function returns the ID of the current [default device](#).

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

#### 5.1.4.45 dhdGetEffectorMass() `int __SDK dhdGetEffectorMass (` `double * mass,` `char ID )`

This function retrieves the mass of the end-effector currently defined for a device. The gripper mass is used in the [gravity compensation](#) feature.

### Parameters

<i>mass</i>	a pointer to the actual end-effector mass in [kg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.46 dhdGetEnc()** `int __SDK dhdGetEnc (`  
    `int enc[DHD_MAX_DOF],`  
    `uchar mask,`  
    `char ID )`

This function retrieves encoder values into encoder array. It is particularly useful when using the the [generic controller](#) directly, without a device model attached.

### Parameters

<i>enc</i>	<b>out</b> encoder values array
<i>mask</i>	<b>[default=0xff]</b> bitwise mask of which encoders should be read in
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Note

**expert mode only - USE AT YOUR OWN RISKS**

### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.47 dhdGetEncoder()** `int __SDK dhdGetEncoder (`  
    `int index,`  
    `char ID )`

This function reads a single encoder value from the haptic device.

### Parameters

<i>index</i>	the encoder index number as defined by <a href="#">DHD_MAX_DOF</a>
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

the (positive) encoder reading on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.48 dhdcGetEncRange()** `int __SDK dhdcGetEncRange (`  
     `int encMin[DHD_MAX_DOF],`  
     `int encMax[DHD_MAX_DOF],`  
     `char ID )`

This function retrieves the expected min and max encoder values for all axis present on the current device. Axis indices that do not exist on the device will return a range of 0.

## Parameters

<i>encMin</i>	<b>out</b> minimum encoder values array
<i>encMax</i>	<b>out</b> maximum encoder values array
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.49 dhdcGetEncVelocities()** `int __SDK dhdcGetEncVelocities (`  
     `double v[DHD_MAX_DOF],`  
     `char ID )`

This function retrieves the encoder angle velocities in [increments/s] for all sensed degrees-of-freedom of the current device.

## Parameters

<i>v</i>	<b>out</b> array of joint angle velocities in [rad/s]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.50 dhdGetForce()** `int __SDK dhdGetForce (`  
`double * fx,`  
`double * fy,`  
`double * fz,`  
`char ID )`

This function retrieves the force vector applied to the end-effector.

### Parameters

<i>fx</i>	<b>[out]</b> force on the X axis in [N]
<i>fy</i>	<b>[out]</b> force on the Y axis in [N]
<i>fz</i>	<b>[out]</b> force on the Z axis in [N]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.51 dhdGetForceAndTorque()** `int __SDK dhdGetForceAndTorque (`  
`double * fx,`  
`double * fy,`  
`double * fz,`  
`double * tx,`  
`double * ty,`  
`double * tz,`  
`char ID )`

This function retrieves the force and torque vectors applied to the device end-effector.

### Parameters

<i>fx</i>	<b>[out]</b> force on the X axis in [N]
<i>fy</i>	<b>[out]</b> force on the Y axis in [N]
<i>fz</i>	<b>[out]</b> force on the Z axis in [N]
<i>tx</i>	<b>[out]</b> torque around the X axis in [Nm]
<i>ty</i>	<b>[out]</b> torque around the Y axis in [Nm]
<i>tz</i>	<b>[out]</b> torque around the Z axis in [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.52 dhdGetForceAndTorqueAndGripperForce()** `int __SDK dhdGetForceAndTorqueAndGripperForce (`  
     `double * fx,`  
     `double * fy,`  
     `double * fz,`  
     `double * tx,`  
     `double * ty,`  
     `double * tz,`  
     `double * f,`  
     `char ID )`

This function retrieves the force and torque vectors applied to the device end-effector, as well as the force applied to the gripper.

**Parameters**

<i>fx</i>	<b>[out]</b> force on the X axis in [N]
<i>fy</i>	<b>[out]</b> force on the Y axis in [N]
<i>fz</i>	<b>[out]</b> force on the Z axis in [N]
<i>tx</i>	<b>[out]</b> torque around the X axis in [Nm]
<i>ty</i>	<b>[out]</b> torque around the Y axis in [Nm]
<i>tz</i>	<b>[out]</b> torque around the Z axis in [Nm]
<i>f</i>	<b>[out]</b> force on the gripper in [N]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.53 dhdGetGripperAngleDeg()** `int __SDK dhdGetGripperAngleDeg (`  
     `double * a,`  
     `char ID )`

This function retrieves the gripper opening angle in degrees.

**Parameters**

<i>a</i>	<b>[out]</b> gripper opening [deg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Note**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

#### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.54 dhdGetGripperAngleRad()** `int __SDK dhdGetGripperAngleRad (`  
    `double * a,`  
    `char ID )`

This function retrieves the gripper opening angle in radians.

#### Parameters

<i>a</i>	<b>[out]</b> gripper opening [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.55 dhdGetGripperAngularVelocityDeg()** `int __SDK dhdGetGripperAngularVelocityDeg (`  
    `double * wg,`  
    `char ID )`

This function retrieves the estimated instantaneous angular velocity of the gripper in [deg/s]. Velocity computation can be configured by calling [dhdConfigGripperVelocity\(\)](#). By default [DHD\\_VELOCITY\\_WINDOW](#) and [DHD\\_VELOCITY\\_WINDOWING](#) are used. See [velocity estimator](#) for details.

## Parameters

<i>wg</i>	<b>[out]</b> gripper angular velocity [deg/s]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## See also

[dhdConfigGripperVelocity\(\)](#) and [velocity estimator](#) for details  
[dhdGetGripperLinearVelocity\(\)](#)  
[dhdGetGripperAngularVelocityRad\(\)](#)

## Note

Please note that the velocity estimator requires at least 2 position updates during the time interval defined in [dhdConfigGripperVelocity\(\)](#) in order to be able to compute the estimate. Otherwise, e.g. if there are no calls to [dhdGetPosition\(\)](#) or [dhdGetGripperAngularVelocityDeg\(\)](#) within the time interval window, [dhdGetGripperAngularVelocityDeg\(\)](#) will return an error ([DHD\\_ERROR\\_TIMEOUT](#)).

## Returns

0 on success, -1 on failure.  
See [error management](#) for details.

**5.1.4.56 dhdGetGripperAngularVelocityRad()** `int __SDK dhdGetGripperAngularVelocityRad (`  
    `double * wg,`  
    `char ID )`

This function retrieves the estimated instantaneous angular velocity of the gripper in [rad/s]. Velocity computation can be configured by calling [dhdConfigGripperVelocity\(\)](#). By default [DHD\\_VELOCITY\\_WINDOW](#) and [DHD\\_VELOCITY\\_WINDOWING](#) are used. See [velocity estimator](#) for details.

## Parameters

<i>wg</i>	<b>[out]</b> gripper angular velocity [rad/s]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## See also

[dhdConfigGripperVelocity\(\)](#) and [velocity estimator](#) for details  
[dhdGetGripperLinearVelocity\(\)](#)  
[dhdGetGripperAngularVelocityDeg\(\)](#)

## Note

Please note that the velocity estimator requires at least 2 position updates during the time interval defined in [dhdConfigGripperVelocity\(\)](#) in order to be able to compute the estimate. Otherwise, e.g. if there are no calls to [dhdGetPosition\(\)](#) or [dhdGetGripperAngularVelocityRad\(\)](#) within the time interval window, [dhdGetGripperAngularVelocityRad\(\)](#) will return an error ([DHD\\_ERROR\\_TIMEOUT](#)).

### Returns

0 on success, -1 on failure.

See [error management](#) for details.

**5.1.4.57 dhdGetGripperEncoder()** `int __SDK dhdGetGripperEncoder (`  
    `int * enc,`  
    `char ID )`

This function retrieves the encoder value of the force gripper.

### Parameters

<i>enc</i>	<b>[out]</b> gripper encoder reading
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Note

**expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

### Returns

0 on success, -1 otherwise.

See [error management](#) for details.

**5.1.4.58 dhdGetGripperFingerPos()** `int __SDK dhdGetGripperFingerPos (`  
    `double * px,`  
    `double * py,`  
    `double * pz,`  
    `char ID )`

This function retrieves the position in Cartesian coordinates of forefinger rest location of the force gripper structure if present.

### Parameters

<i>px</i>	<b>[out]</b> gripper finger X coord
<i>py</i>	<b>[out]</b> gripper finger Y coord
<i>pz</i>	<b>[out]</b> gripper finger Z coord
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)



**Note**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)

**Returns**

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.59 dhcGetGripperGap()** `int __SDK dhcGetGripperGap (`  
     `double * g,`  
     `char ID )`

This function retrieves the gripper opening distance in meters.

**Parameters**

<i>g</i>	<b>[out]</b> gripper opening [m]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Note**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

**Returns**

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.60 dhcGetGripperLinearVelocity()** `int __SDK dhcGetGripperLinearVelocity (`  
     `double * vg,`  
     `char ID )`

This function retrieves the estimated instantaneous linear velocity of the gripper in [m/s]. Velocity computation can be configured by calling [dhcConfigGripperVelocity\(\)](#). By default [DHD\\_VELOCITY\\_WINDOW](#) and [DHD\\_VELOCITY\\_WINDOWING](#) are used. See [velocity estimator](#) for details.

## Parameters

<i>vg</i>	<b>[out]</b> gripper linear velocity [m/s]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## See also

[dhdConfigGripperVelocity\(\)](#) and [velocity estimator](#) for details  
[dhdGetGripperAngularVelocityRad\(\)](#)  
[dhdGetGripperAngularVelocityDeg\(\)](#)

## Note

Please note that the velocity estimator requires at least 2 position updates during the time interval defined in [dhdConfigGripperVelocity\(\)](#) in order to be able to compute the estimate. Otherwise, e.g. if there are no calls to [dhdGetPosition\(\)](#) or [dhdGetGripperLinearVelocity\(\)](#) within the time interval window, [dhdGetGripperLinearVelocity\(\)](#) will return an error ([DHD\\_ERROR\\_TIMEOUT](#)).

## Returns

0 on success, -1 on failure.  
 See [error management](#) for details.

**5.1.4.61 dhdGetGripperThumbPos()** `int __SDK dhdGetGripperThumbPos (`  
     `double * px,`  
     `double * py,`  
     `double * pz,`  
     `char ID )`

This function retrieves the position in Cartesian coordinates of thumb rest location of the force gripper structure if present.

## Parameters

<i>px</i>	<b>[out]</b> gripper thumb X coord
<i>py</i>	<b>[out]</b> gripper thumb Y coord
<i>pz</i>	<b>[out]</b> gripper thumb Z coord
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)

## Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.62 dhdGetJointAngleRange()** `int __SDK dhdGetJointAngleRange (`  
`double jmin[DHD_MAX_DOF],`  
`double jmax[DHD_MAX_DOF],`  
`char ID )`

This function retrieves the expected min and max joint angles in [rad] for all sensed degrees-of-freedom on the current device. Axis indices that do not exist on the device will return a range of 0.0.

## Parameters

<i>jmin</i>	<b>outarray</b> of min joint angles in [rad]
<i>jmax</i>	<b>outarray</b> of max joint angles in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.63 dhdGetJointAngles()** `int __SDK dhdGetJointAngles (`  
`double j[DHD_MAX_DOF],`  
`char ID )`

This function retrieves the joint angles in [rad] for all sensed degrees-of-freedom of the current device.

## Parameters

<i>j</i>	<b>outarray</b> of joint angles in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.64 dhdGetJointVelocities()** `int __SDK dhdGetJointVelocities (`  
`double v[DHD_MAX_DOF],`  
`char ID )`

This function retrieves the joint angle velocities in [rad/s] for all sensed degrees-of-freedom of the current device.

#### Parameters

<i>v</i>	<b>out</b> array of joint angle velocities in [rad/s]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

#### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.65 dhdGetLinearVelocity()** `int __SDK dhdGetLinearVelocity (`  
`double * vx,`  
`double * vy,`  
`double * vz,`  
`char ID )`

This function retrieves the estimated instantaneous translational velocity. Velocity computation can be configured by calling [dhdConfigLinearVelocity\(\)](#). By default [DHD\\_VELOCITY\\_WINDOW](#) and [DHD\\_VELOCITY\\_WINDOWING](#) are used. See [velocity estimator](#) for details.

#### Parameters

<i>vx</i>	<b>[out]</b> velocity along the X axis [m/s]
<i>vy</i>	<b>[out]</b> velocity along the Y axis [m/s]
<i>vz</i>	<b>[out]</b> velocity along the Z axis [m/s]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### See also

[dhdConfigLinearVelocity\(\)](#) and [velocity estimator](#) for details

#### Note

Please note that the velocity estimator requires at least 2 position updates during the time interval defined in [dhdConfigLinearVelocity\(\)](#) in order to be able to compute the estimate. Otherwise, e.g. if there are no calls to [dhdGetPosition\(\)](#) or [dhdGetLinearVelocity\(\)](#) within the time interval window, [dhdGetLinearVelocity\(\)](#) will return an error ([DHD\\_ERROR\\_TIMEOUT](#)).

#### Returns

0 on success, -1 on failure.  
 See [error management](#) for details.

**5.1.4.66 dhdGetMaxForce()** `double __SDK dhdGetMaxForce (`  
`char ID )`

This function retrieves the current limit (in N) to the force magnitude that can be applied by the haptic device. If the return value is negative, the limit is disabled and the full range of force available can be applied.

#### Parameters

<i>ID</i>	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)
-----------	---

#### See also

[dhdSetMaxForce\(\)](#)  
[dhdGetMaxTorque\(\)](#)  
[dhdGetMaxGripperForce\(\)](#)  
[dhdGetMaxPower\(\)](#)  
[dhdGetMaxUsablePower\(\)](#)

#### Returns

The current force limit (in N) if set, -1.0 if no limit is enforced.

**5.1.4.67 dhdGetMaxGripperForce()** `double __SDK dhdGetMaxGripperForce (`  
`char ID )`

This function retrieves the current limit (in N) to the force magnitude that can be applied by the haptic device gripper. If the return value is negative, the limit is disabled and the full range of gripper force available can be applied.

#### Parameters

<i>ID</i>	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)
-----------	---

#### See also

[dhdSetMaxGripperForce\(\)](#)  
[dhdGetMaxForce\(\)](#)  
[dhdGetMaxTorque\(\)](#)  
[dhdGetMaxPower\(\)](#)  
[dhdGetMaxUsablePower\(\)](#)

#### Returns

The current gripper force limit (in N) if set, -1.0 if no limit is enforced.

**5.1.4.68 dhdGetMaxTorque()** `double __SDK dhdGetMaxTorque (`  
`char ID )`

This function retrieves the current limit (in Nm) to the torque magnitude that can be applied by the haptic device. If the return value is negative, the limit is disabled and the full range of torque available can be applied.

## Parameters

<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

## See also

[dhdSetMaxTorque\(\)](#)  
[dhdGetMaxForce\(\)](#)  
[dhdGetMaxGripperForce\(\)](#)  
[dhdGetMaxPower\(\)](#)  
[dhdGetMaxUsablePower\(\)](#)

## Returns

The current torque limit (in Nm) if set, -1.0 if no limit is enforced.

**5.1.4.69 dhdGetOrientationDeg()** `int __SDK dhdGetOrientationDeg (`  
     `double * oa,`  
     `double * ob,`  
     `double * og,`  
     `char ID )`

For devices with a wrist structure, This function retrieves individual angle of each joint, starting with the one located nearest to the wrist base plate. For the [DHD\\_DEVICE\\_OMEGA33](#) and [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#) devices, angles are computed with respect to their internal reference frame, which is rotated 45 degrees around the Y axis. Please refer to your device user manual for more information on your device coordinate system.

## Parameters

<i>oa</i>	<b>[out]</b> device orientation around the first wrist joint in [deg]
<i>ob</i>	<b>[out]</b> device orientation around the second wrist joint in [deg]
<i>og</i>	<b>[out]</b> device orientation around the third wrist joint in [deg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA33](#)
- [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

## Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

## See also

[dhdGetOrientationRad\(\)](#)

**5.1.4.70 dhdGetOrientationFrame()** `int __SDK dhdGetOrientationFrame (`  
`double matrix[3][3],`  
`char ID )`

This function retrieves the rotation matrix of the wrist structure. The identity matrix is returned for devices that do not support orientations.

## Parameters

<i>matrix</i>	<b>[out]</b> orientation matrix frame
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.71 dhdGetOrientationRad()** `int __SDK dhdGetOrientationRad (`  
`double * oa,`  
`double * ob,`  
`double * og,`  
`char ID )`

For devices with a wrist structure, this function retrieves individual angle of each joint, starting with the one located nearest to the wrist base plate. For the [DHD\\_DEVICE\\_OMEGA33](#) and [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#) devices, angles are computed with respect to their internal reference frame, which is rotated 45 degrees around the Y axis. Please refer to your device user manual for more information on your device coordinate system.

## Parameters

<i>oa</i>	<b>[out]</b> device orientation around the first wrist joint in [rad]
<i>ob</i>	<b>[out]</b> device orientation around the second wrist joint in [rad]
<i>og</i>	<b>[out]</b> device orientation around the third wrist joint in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA33](#)

- [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

#### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

#### See also

[dhdGetOrientationDeg\(\)](#)

**5.1.4.72 dhdGetPosition()** `int __SDK dhdGetPosition (`  
     `double * px,`  
     `double * py,`  
     `double * pz,`  
     `char ID )`

This function retrieves the position of the end-effector in Cartesian coordinates. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>px</i>	<b>[out]</b> device position on the X axis in [m]
<i>py</i>	<b>[out]</b> device position on the Y axis in [m]
<i>pz</i>	<b>[out]</b> device position on the Z axis in [m]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

#### Examples

[hello\\_world.cpp](#).

**5.1.4.73 dhdGetPositionAndOrientationDeg()** `int __SDK dhdGetPositionAndOrientationDeg (`  
     `double * px,`  
     `double * py,`  
     `double * pz,`



```
double * oa,
double * ob,
double * og,
char ID )
```

This function retrieves the position and orientation of the end-effector in Cartesian coordinates. For devices with a wrist structure, the orientation is expressed as the individual angle of each joint, starting with the one located nearest to the wrist base plate. For the [DHD\\_DEVICE\\_OMEGA33](#) and [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#) devices, angles are computed with respect to their internal reference frame, which is rotated 45 degrees around the Y axis. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>px</i>	<b>[out]</b> device position on the X axis in [m]
<i>py</i>	<b>[out]</b> device position on the Y axis in [m]
<i>pz</i>	<b>[out]</b> device position on the Z axis in [m]
<i>oa</i>	<b>[out]</b> device orientation around the first wrist joint in [deg]
<i>ob</i>	<b>[out]</b> device orientation around the second wrist joint in [deg]
<i>og</i>	<b>[out]</b> device orientation around the thrid wrist joint in [deg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

#### 5.1.4.74 dhdcGetPositionAndOrientationFrame() `int __SDK dhdcGetPositionAndOrientationFrame (`

```
double * px,
double * py,
double * pz,
double matrix[3][3],
char ID )
```

This function retrieves the position and orientation matrix of the end-effector in Cartesian coordinates. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>px</i>	<b>[out]</b> device position on the X axis in [m]
<i>py</i>	<b>[out]</b> device position on the Y axis in [m]
<i>pz</i>	<b>[out]</b> device position on the Z axis in [m]
<i>matrix</i>	<b>[out]</b> orientation matrix frame
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.75 dhdGetPositionAndOrientationRad()** `int __SDK dhdGetPositionAndOrientationRad (`  
`double * px,`  
`double * py,`  
`double * pz,`  
`double * oa,`  
`double * ob,`  
`double * og,`  
`char ID )`

This function retrieves the position and orientation of the end-effector in Cartesian coordinates. For devices with a wrist structure, the orientation is expressed as the individual angle of each joint, starting with the one located nearest to the wrist base plate. For the [DHD\\_DEVICE\\_OMEGA33](#) and [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#) devices, angles are computed with respect to their internal reference frame, which is rotated 45 degrees around the Y axis. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>px</i>	<b>[out]</b> device position on the X axis in [m]
<i>py</i>	<b>[out]</b> device position on the Y axis in [m]
<i>pz</i>	<b>[out]</b> device position on the Z axis in [m]
<i>oa</i>	<b>[out]</b> device orientation around the first wrist joint in [rad]
<i>ob</i>	<b>[out]</b> device orientation around the second wrist joint in [rad]
<i>og</i>	<b>[out]</b> device orientation around the third wrist joint in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.76 dhdGetSDKVersion()** `void __SDK dhdGetSDKVersion (`  
`int * major,`  
`int * minor,`  
`int * release,`  
`int * revision )`

This function returns the SDK complete set of version numbers.

#### Parameters

<i>major</i>	major version number
<i>minor</i>	minor version number
<i>release</i>	release number
<i>revision</i>	revision number

#### See also

[dhdGetSDKVersionStr\(\)](#)

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.77 dhdGetSDKVersionStr()** `const char* __SDK dhdGetSDKVersionStr ( )`

This function returns a string that fully describes the SDK version.

**Returns**

The version string on success, an empty string otherwise.

**5.1.4.78 dhdGetSerialNumber()** `int __SDK dhdGetSerialNumber (`  
`ushort * sn,`  
`char ID )`

This function returns the device serial number.

**Parameters**

<i>sn</i>	<b>[out]</b> a pointer to a valid <code>unsigned char</code> to receive the device serial number
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.79 dhdGetStatus()** `int __SDK dhdGetStatus (`  
`int status[DHD_MAX_STATUS],`  
`char ID )`

This function returns the status vector of the haptic device. The status is described in the [status](#) section.

**Parameters**

<i>status</i>	<b>[out]</b> an array that will receive the status vector
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.80 dhdGetSystemCounter()** `ulong __SDK dhdGetSystemCounter ( )`

This function returns a timestamp computed from the high-resolution system counter, expressed in microseconds. This function is deprecated, please use [dhdGetTime\(\)](#) instead.

**Deprecated** This function is deprecated and is kept for backward compatibility only.

**Returns**

A timestamp in [us].

**See also**

See [dhdGetTime\(\)](#)

**5.1.4.81 dhdGetSystemName()** `const char* __SDK dhdGetSystemName (   
char ID )`

This function returns the haptic device [type](#). As this SDK can be used to control all of Force Dimension haptic products, this can help programmers ensure that their application is running on the appropriate target haptic device.

**Parameters**

<i>ID</i>	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)
-----------	---

**Returns**

The [device type](#) string on success, NULL otherwise.  
See [error management](#) for details.

**5.1.4.82 dhdGetSystemRev()** `int __SDK dhdGetSystemRev (   
char ID )`

This function returns the revision associated with this instance of haptic device [type](#). As this SDK can be used to control all of Force Dimension haptic products, this can help programmers ensure that their application is running on the appropriate target haptic device.

**Parameters**

<i>ID</i>	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)
-----------	---

**Returns**

The device type revision on success, -1 otherwise.  
See [error management](#) for details.

5.1.4.83 dhdGetSystemType() int \_\_SDK dhdGetSystemType ( char ID )

This function returns the haptic device [type](#). As this SDK can be used to control all of Force Dimension haptic products, this can help programmers ensure that their application is running on the appropriate target haptic device.

Parameters

ID	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)
----	---

Returns

The [device type](#) on success, -1 otherwise.  
See [error management](#) for details.

5.1.4.84 dhdGetTime() double \_\_SDK dhdGetTime ( )

This function returns the current value from the high-resolution system counter in [s]. The resolution of the system counter may be machine-dependent, as it is usually derived from one of the CPU clocks signals. The time returned is guaranteed to be monotonic.

Note

On QNX platforms, the time granularity is defined by the system tick resolution. Please take a look at QNX Clock↔Period() documentation for details.

Returns

The current time in [s].

5.1.4.85 dhdGetVelocityThreshold() int \_\_SDK dhdGetVelocityThreshold ( uint \* val, char ID )

This function retrieves the current [velocity threshold](#) of the device. Velocity threshold is a safety feature that prevents the device from accelerating to high velocities without control. If the velocity of one of the device axis passes the threshold, the device enters [BRAKES mode](#).

Parameters

val	an arbitrary value of velocity threshold the range of threshold values is device dependent, it is recommended <b>NOT</b> to modify factory settings
ID	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)

Note

[expert mode](#) only - USE AT YOUR OWN RISKS

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.86 dhbGetVersion()** `int __SDK dhbGetVersion (`  
     `double * ver,`  
     `char ID )`

This function returns the [device controller](#) version. As this SDK can be used to control all of Force Dimension haptic products, this can help programmers ensure that their application is running on the appropriate version of the haptic controller.

**Parameters**

<i>ver</i>	<b>[out]</b> pointer to a variable that will receive the controller release version number
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.87 dhbGetVersionStr()** `int __SDK dhbGetVersionStr (`  
     `char * buffer,`  
     `size_t size,`  
     `char ID )`

This function returns the [device controller](#) version string. As this SDK can be used to control all of Force Dimension haptic products, this can help programmers ensure that their application is running on the appropriate version of the haptic controller.

**Parameters**

<i>buffer</i>	A user-allocated buffer to receive the device controller version string.
<i>size</i>	The size of the user-allocated version string buffer.
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.88 dhbGetWatchdog()** `int __SDK dhbGetWatchdog (`  
     `uchar * val,`  
     `char ID )`

This function retrieves the watchdog duration in multiples of 125 microseconds on compatible devices.

## Parameters

<i>val</i>	<b>[out]</b> watchdog duration in multiples of 125 [us]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## See also

[dhdSetWatchdog\(\)](#)

## Returns

0 on success, -1 otherwise.

See [error management](#) for details.

**5.1.4.89 dhdcGetWristEncoders()** `int __SDK dhdcGetWristEncoders (`  
    `int * enc0,`  
    `int * enc1,`  
    `int * enc2,`  
    `char ID )`

This function reads the encoders values of the wrist structure.

## Parameters

<i>enc0</i>	<b>[out]</b> wrist joint 0 encoder reading
<i>enc1</i>	<b>[out]</b> wrist joint 1 encoder reading
<i>enc2</i>	<b>[out]</b> wrist joint 2 encoder reading
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA33](#)
- [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.90 dhdGetWristJacobian()** `int __SDK dhdGetWristJacobian (`  
    `double jacobian[3][3],`  
    `char ID )`

This function retrieves the wrist jacobian matrix based on the current end-effector position. Please refer to your device user manual for more information on your device coordinate system.

### Parameters

<i>jacobian</i>	<b>[out]</b> device jacobian
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.91 dhdGetWristJointAngles()** `int __SDK dhdGetWristJointAngles (`  
    `double * j0,`  
    `double * j1,`  
    `double * j2,`  
    `char ID )`

This function retrieves the joint angles in [rad] for the wrist structure.

### Parameters

<i>j0</i>	<b>[out]</b> joint angle for joint 0 in [rad]
<i>j1</i>	<b>[out]</b> joint angle for joint 1 in [rad]
<i>j2</i>	<b>[out]</b> joint angle for joint 2 in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Note

**[expert mode](#) only - USE AT YOUR OWN RISKS**

### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.



**5.1.4.92 dhdGripperAngleRadToEncoder()** `int __SDK dhdGripperAngleRadToEncoder (`  
    `double a,`  
    `int * enc,`  
    `char ID )`

This function computes and returns the gripper encoder value for a given gripper angle in [rad]

#### Parameters

<i>a</i>	gripper opening in [rad]
<i>enc</i>	<b>[out]</b> gripper encoder reading
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

#### Returns

0 on success, -1 otherwise.

See [error management](#) for details.

**5.1.4.93 dhdGripperEncoderToAngleRad()** `int __SDK dhdGripperEncoderToAngleRad (`  
    `int enc,`  
    `double * a,`  
    `char ID )`

This function computes and returns the opening of the gripper as an angle in [rad] for a given encoder reading.

#### Parameters

<i>enc</i>	gripper encoder reading
<i>a</i>	<b>[out]</b> gripper opening [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)

- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

#### Returns

0 or [DHD\\_TIMEGUARD](#) on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.94 dhdGripperEncoderToGap()** `int __SDK dhdGripperEncoderToGap (`  
`int enc,`  
`double * gap,`  
`char ID )`

This function computes and returns the opening of the gripper as a distance in [m] for a given encoder reading.

#### Parameters

<i>enc</i>	gripper encoder reading
<i>gap</i>	<b>[out]</b> gripper opening [m]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

##### **expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.95 dhdGripperForceToMotor()** `int __SDK dhdGripperForceToMotor (`  
`double frc,`  
`ushort * mot,`  
`int enc[4],`  
`char ID )`

Given a desired force to be displayed by the force gripper, this function computes and returns the corresponding motor command.

## Parameters

<i>frc</i>	force on the gripper end-effector [N]
<i>mot</i>	<b>[out]</b> motor command on gripper axis
<i>enc</i>	encoder reading for wrist (at indices 0,1,2) and gripper (at index 3)
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

## Returns

0 or [DHD\\_MOTOR\\_SATURATED](#) on success, -1 otherwise.

See [error management](#) for details.

**5.1.4.96 dhgGripperGapToEncoder()** `int __SDK dhgGripperGapToEncoder (`  
     double *gap*,  
     int \* *enc*,  
     char *ID* )

This function computes and returns the gripper encoder value for a given gripper opening distance in [m]

## Parameters

<i>gap</i>	gripper opening in [m]
<i>enc</i>	<b>[out]</b> gripper encoder reading
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.97 dhdGripperMotorToForce()** `int __SDK dhdGripperMotorToForce (`  
    `ushort mot,`  
    `double * frc,`  
    `int enc[4],`  
    `char ID )`

This function computes and returns the force applied to the end-effector for a given motor command.

### Parameters

<i>mot</i>	motor command on gripper axis
<i>frc</i>	<b>[out]</b> force on the gripper end-effector [N]
<i>enc</i>	encoder reading for wrist (at indices 0,1,2) and gripper (at index 3)
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Note

#### **expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.98 dhdHasActiveGripper()** `bool __SDK dhdHasActiveGripper (`  
    `char ID )`

This function returns **true** if the device has an active gripper, **false** otherwise.

### Parameters

<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

**Note**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

**Returns**

**true** if the device is configure for left-handed use, **false** otherwise.

**5.1.4.99 dhdHasActiveWrist()** `bool __SDK dhdHasActiveWrist (`  
    `char ID )`

This function returns **true** if the device has an active wrist, **false** otherwise.

**Parameters**

<i>ID</i>	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)
-----------	---

**Note**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA33](#)
- [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

**Returns**

**true** if the device is configure for left-handed use, **false** otherwise.

**5.1.4.100 dhdHasBase()** `bool __SDK dhdHasBase (`  
    `char ID )`

This function returns **true** if the device has a base, **false** otherwise.

#### Parameters

<i>ID</i>	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)
-----------	---

#### Note

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_DELTA3](#)
- [DHD\\_DEVICE\\_OMEGA3](#)
- [DHD\\_DEVICE\\_OMEGA33](#)
- [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)
- [DHD\\_DEVICE\\_FALCON](#)

#### Returns

**true** if the device is configure for left-handed use, **false** otherwise.

**5.1.4.101 dhdHasGripper()** `bool __SDK dhdHasGripper (`  
`char ID )`

This function returns **true** if the device has a gripper, **false** otherwise.

#### Parameters

<i>ID</i>	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)
-----------	---

#### Note

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

#### Returns

**true** if the device is configure for left-handed use, **false** otherwise.

**5.1.4.102 dhdHasWrist()** `bool __SDK dhdHasWrist (`  
`char ID )`

This function returns **true** if the device has a wrist, **false** otherwise.

#### Parameters

<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

#### Note

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA33](#)
- [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

#### Returns

**true** if the device is configure for left-handed use, **false** otherwise.

**5.1.4.103 dhdIsLeftHanded()** `bool __SDK dhdIsLeftHanded (`  
`char ID )`

This function returns **true** if the device is configured for left-handed use, **false** otherwise.

#### Parameters

<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

#### Note

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA33](#)
- [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

## Returns

**true** if the device is configured for left-handed use, **false** otherwise.

**5.1.4.104 dhdJointAnglesToInertiaMatrix()** `int __SDK dhdJointAnglesToInertiaMatrix (`  
`double j[DHD_MAX_DOF],`  
`double inertia[6][6],`  
`char ID )`

This function retrieves the (Cartesian) inertia matrix based on a given joint configuration. Please refer to your device user manual for more information on your device coordinate system.

## Parameters

<i>j</i>	array of joint angles in [rad]
<i>inertia</i>	<b>[out]</b> device inertia matrix
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## See also

[dhdGetJointAngles\(\)](#)

## Returns

0 on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.105 dhdKbGet()** `char __SDK dhdKbGet ( )`

This function retrieves a character from the keyboard (OS independent).

## Returns

The matching keyboard character.

**5.1.4.106 dhdKbHit()** `bool __SDK dhdKbHit ( )`

This function checks the keyboard for a key hit (OS independent).

## Returns

**true** on key pressed, false otherwise.



**5.1.4.107 dhdcOpen()** `int __SDK dhdcOpen ( )`

This function opens a connection to the first available device connected to the system. The order in which devices are opened persists until devices are added or removed.

**Note**

If this call is successful, the [default device ID](#) is set to the newly opened device. See the [multiple device](#) section for more information on using multiple devices on the same computer.

**Returns**

The device ID on success, -1 otherwise.  
See [error management](#) for details.

**See also**

[dhdcOpenID\(\)](#)

**Examples**

[hello\\_world.cpp](#), and [single\\_device.cpp](#).

**5.1.4.108 dhdcOpenID()** `int __SDK dhdcOpenID (`  
`char index )`

This function opens a connection to one particular device connected to the system. The order in which devices are opened persists until devices are added or removed. If the device at the specified index is already opened, its device ID is returned.

**Parameters**

<i>index</i>	the device enumeration index, as assigned by the underlying operating system (must be between 0 and the number of devices connected to the system)
--------------	--

**Note**

If this call is successful, the [default device ID](#) is set to the newly opened device. See the [multiple device](#) section for more information on using multiple devices on the same computer.

**Returns**

The device ID on success, -1 otherwise.  
See [error management](#) for details.

**See also**

[dhdcOpen\(\)](#)

**Examples**

[multiple\\_devices.cpp](#).

**5.1.4.109 dhdOpenSerial()** `int __SDK dhdOpenSerial (`  
`int serial )`

This function opens a connection to the device with a given serial number (available on recent models only).

**Parameters**

<i>serial</i>	requested system serial number
---------------	--------------------------------

**Note**

If this call is successful, the [default device ID](#) is set to the newly opened device. See the [multiple device](#) section for more information on using multiple devices on the same computer.

**Returns**

The device ID on success, -1 otherwise.  
See [error management](#) for details.

**See also**

[dhdOpenID\(\)](#)

**5.1.4.110 dhdOpenType()** `int __SDK dhdOpenType (`  
`int type )`

This function opens a connection to the first device of a given type connected to the system. The order in which devices are opened persists until devices are added or removed.

**Parameters**

<i>type</i>	requested system <a href="#">Device Types</a> type
-------------	--

**Note**

If this call is successful, the [default device ID](#) is set to the newly opened device. See the [multiple device](#) section for more information on using multiple devices on the same computer.

**Returns**

The device ID on success, -1 otherwise.  
See [error management](#) for details.

**See also**

[dhdOpenID\(\)](#)

**5.1.4.111 dhdpReloadMot()** `int __SDK dhdpReloadMot (`  
    `ushort mot[DHD_MAX_DOF],`  
    `uchar mask,`  
    `char ID )`

This function programs motor commands to a selection of motor channels. Unlike [dhdpSetMot\(\)](#), this function saves the requested commands internally for later application by calling [dhdpSetForce\(\)](#) and the likes.

#### Parameters

<i>mot</i>	motor values array
<i>mask</i>	<b>[default=0xff]</b> bitwise mask of which motor should be set
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.112 dhdpPreset()** `int __SDK dhdpPreset (`  
    `int val[DHD_MAX_DOF],`  
    `uchar mask,`  
    `char ID )`

This function sets selected encoder offsets to a given value. Intended for use with the the [generic controller](#) when no RESET button is available.

#### Parameters

<i>val</i>	motor values array
<i>mask</i>	bitwise mask of which encoder should be set
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.113 dhdReadConfigFromFile()** `int __SDK dhdReadConfigFromFile (`  
    `char * filename,`  
    `char ID )`

This function loads a specific device calibration/configuration data from a file. Particularly useful when using the [generic controller](#) connected to a Force Dimension device without using the [dhdControllerSetDevice\(\)](#) call.

## Parameters

<i>filename</i>	configuration file
<i>ID</i>	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.114 dhdcReset()** `int __SDK dhdcReset (`  
    `char ID )`

This function puts the device in [RESET mode](#).

## Parameters

<i>ID</i>	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)
-----------	---

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.115 dhdcResetWrist()** `int __SDK dhdcResetWrist (`  
    `char ID )`

This function resets the wrist calibration on a **delta.6** haptic device.

## Parameters

<i>ID</i>	[default=-1] device ID (see <a href="#">multiple devices</a> section for details)
-----------	---

**Deprecated** This function is deprecated and is kept for backward compatibility only.

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.116 dhdSetBaseAngleXDeg()** `int __SDK dhdSetBaseAngleXDeg (`  
    `double angle,`  
    `char ID )`

This function sets the device base plate angle around the X axis. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>angle</i>	device base plate angle around X [deg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

#### See also

[dhdSetBaseAngleXRad\(\)](#)

**5.1.4.117 dhdSetBaseAngleXRad()** `int __SDK dhdSetBaseAngleXRad (`  
    `double angle,`  
    `char ID )`

This function sets the device base plate angle around the X axis. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>angle</i>	device base plate angle around X [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

#### See also

[dhdSetBaseAngleXDeg\(\)](#)

**5.1.4.118 dhdSetBaseAngleZDeg()** `int __SDK dhdSetBaseAngleZDeg (`  
    `double angle,`  
    `char ID )`

This function sets the device base plate angle around the vertical Z axis. Please refer to your device user manual for more information on your device coordinate system.

## Parameters

<i>angle</i>	device base plate angle around Z [deg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

## See also

[dhdSetBaseAngleZRad\(\)](#)

**5.1.4.119 dhdSetBaseAngleZRad()** `int __SDK dhdSetBaseAngleZRad (`  
    `double angle,`  
    `char ID )`

This function sets the device base plate angle around the vertical Z axis. Please refer to your device user manual for more information on your device coordinate system.

## Parameters

<i>angle</i>	device base plate angle around Z [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

## See also

[dhdSetBaseAngleZDeg\(\)](#)

**5.1.4.120 dhdSetBrakes()** `int __SDK dhdSetBrakes (`  
    `int val,`  
    `char ID )`

This function toggles the device [electromagnetic brakes](#) state.

## Parameters

<i>val</i>	desired state of the brakes (DHD_ON or DHD_OFF)
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.121 dhdSetBrk()** `int __SDK dhdSetBrk (`  
     `uchar mask,`  
     `char ID )`

This function sets brakes [electromagnetic brakes](#) status on selective motor groups. Only applies when using the [generic controller](#) directly, without a device model attached.

## Parameters

<i>mask</i>	<b>[default=0xff]</b> bitwise mask of which motor group should the brakes be set on.
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

### **expert mode only - USE AT YOUR OWN RISKS**

The motors on the [dhd\\_controller](#) are grouped as follows:

- group1 - [mot0,mot1,mot2]
- group2 - [mot3,mot4,mot5]
- group3 - [mot6]
- group4 - [mot7]

The *mask* argument addresses all 8 motors bitwise. If a single bit within a motor group address is enabled, the entire motor group [dhd\\_brakes](#) will be activated.

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.122 dhdSetComMode()** `int __SDK dhdSetComMode (`  
     `int mode,`  
     `char ID )`

This function sets the [COM operation mode](#) on compatible devices.

## Parameters

<i>mode</i>	desired <a href="#">COM operation mode</a>
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)



Note

**expert mode only - USE AT YOUR OWN RISKS**

Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

See also

[dhdSetComModePriority\(\)](#)

**5.1.4.123 dhdSetComModePriority()** `int __SDK dhdSetComModePriority (
 int priority,
 char ID )`

This function sets the priority of the thread (if any) that supports the current [COM operation mode](#) on compatible devices. Currently unused.

Parameters

<i>priority</i>	desired thread priority
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

Note

**expert mode only - USE AT YOUR OWN RISKS**

Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

See also

[dhdSetComMode\(\)](#)

**5.1.4.124 dhdSetDeltaJointTorques()** `int __SDK dhdSetDeltaJointTorques (
 double t0,
 double t1,
 double t2,
 char ID )`

This function sets all joint torques of the DELTA structure.

## Parameters

<i>t0</i>	DELTA axis 0 torque command [Nm]
<i>t1</i>	DELTA axis 1 torque command [Nm]
<i>t2</i>	DELTA axis 2 torque command [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

```
5.1.4.125 dhdSetDeltaMotor()  int __SDK dhdSetDeltaMotor (
    ushort mot0,
    ushort mot1,
    ushort mot2,
    char ID )
```

This function sets desired motor commands to the amplifier channels commanding the DELTA motors.

## Parameters

<i>mot0</i>	DELTA axis 0 motor command
<i>mot1</i>	DELTA axis 1 motor command
<i>mot2</i>	DELTA axis 2 motor command
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

```
5.1.4.126 dhdSetDevice()  int __SDK dhdSetDevice (
    char ID )
```

This function selects the [default device](#) that will receive the SDK commands. The SDK supports [multiple devices](#). This routine allows the programmer to decide which device the SDK [dhd\\_single\\_device\\_call](#) single-device calls will address. Any subsequent SDK call that does not specifically mention the device ID in its parameter list will be sent to that device.

## Parameters

<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

## Returns

0 on success, -1 otherwise.

See [error management](#) for details.

**5.1.4.127 dhdcSetDeviceAngleDeg()** `int __SDK dhdcSetDeviceAngleDeg (`  
    `double angle,`  
    `char ID )`

This function sets the device base plate angle around the (inverted) Y axis. Please refer to your device user manual for more information on your device coordinate system. An angle value of 0 corresponds to the device "upright" position, with its base plate perpendicular to axis X. An angle value of 90 corresponds to the device base plate resting horizontally.

## Parameters

<i>angle</i>	device base plate angle [deg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

0 on success, -1 otherwise.

See [error management](#) for details.

## See also

[dhdcSetDeviceAngleRad\(\)](#)

**5.1.4.128 dhdcSetDeviceAngleRad()** `int __SDK dhdcSetDeviceAngleRad (`  
    `double angle,`  
    `char ID )`

This function sets the device base plate angle around the (inverted) Y axis. Please refer to your device user manual for more information on your device coordinate system. An angle value of 0 corresponds to the device "upright" position, with its base plate perpendicular to axis X. An angle value of Pi/2 corresponds to the device base plate resting horizontally.

## Parameters

<i>angle</i>	device base plate angle [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

### See also

[dhdSetDeviceAngleDeg\(\)](#)

**5.1.4.129 dhdSetEffectorMass()** `int __SDK dhdSetEffectorMass (`  
    `double mass,`  
    `char ID )`

This function defines the mass of the end-effector. This function is required to provide accurate [gravity compensation](#) when custom-made or modified end-effectors are used.

### Parameters

<i>mass</i>	the actual end-effector mass in [kg]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.130 dhdSetForce()** `int __SDK dhdSetForce (`  
    `double fx,`  
    `double fy,`  
    `double fz,`  
    `char ID )`

This function sets the desired force vector in Cartesian coordinates to be applied to the end-effector of the device.

### Parameters

<i>fx</i>	force on the X axis in [N]
<i>fy</i>	force on the Y axis in [N]
<i>fz</i>	force on the Z axis in [N]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Returns

0 or [DHD\\_MOTOR\\_SATURATED](#) on success, -1 otherwise.  
See [error management](#) for details.

### Examples

[hello\\_world.cpp](#), [multiple\\_devices.cpp](#), and [single\\_device.cpp](#).

**5.1.4.131 dhdcSetForceAndGripperForce()** `int __SDK dhdcSetForceAndGripperForce (`  
     `double fx,`  
     `double fy,`  
     `double fz,`  
     `double fg,`  
     `char ID )`

This function sets the desired force vector in Cartesian coordinates and the desired grasping force to be applied to the device end-effector and force gripper.

#### Parameters

<i>fx</i>	translation force along X axis
<i>fy</i>	translation force along Y axis
<i>fz</i>	translation force along Z axis
<i>fg</i>	grasping force
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 or [DHD\\_MOTOR\\_SATURATED](#) on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.132 dhdcSetForceAndTorque()** `int __SDK dhdcSetForceAndTorque (`  
     `double fx,`  
     `double fy,`  
     `double fz,`  
     `double tx,`  
     `double ty,`  
     `double tz,`  
     `char ID )`

This function sets the desired force and torque vectors to be applied to the device end-effector.

#### Parameters

<i>fx</i>	force on the X axis in [N]
<i>fy</i>	force on the Y axis in [N]
<i>fz</i>	force on the Z axis in [N]
<i>tx</i>	torque around the X axis in [Nm]
<i>ty</i>	torque around the Y axis in [Nm]
<i>tz</i>	torque around the Z axis in [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 or [DHD\\_MOTOR\\_SATURATED](#) on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.133 dhdSetForceAndTorqueAndGripperForce()** `int __SDK dhdSetForceAndTorqueAndGripperForce (`  
`double fx,`  
`double fy,`  
`double fz,`  
`double tx,`  
`double ty,`  
`double tz,`  
`double fg,`  
`char ID )`

This function sets the desired force and torque vectors in Cartesian coordinates and the desired grasping force to be applied to the device end-effector and force gripper.

#### Parameters

<i>fx</i>	force on the X axis in [N]
<i>fy</i>	force on the Y axis in [N]
<i>fz</i>	force on the Z axis in [N]
<i>tx</i>	torque around the X axis in [Nm]
<i>ty</i>	torque around the Y axis in [Nm]
<i>tz</i>	torque around the Z axis in [Nm]
<i>fg</i>	gripper force in [N]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 or [DHD\\_MOTOR\\_SATURATED](#) on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.134 dhdSetForceAndWristJointTorques()** `int __SDK dhdSetForceAndWristJointTorques (`  
`double fx,`  
`double fy,`  
`double fz,`  
`double t0,`  
`double t1,`  
`double t2,`  
`char ID )`

This function sets Cartesian force and wrist joint torques.

#### Parameters

<i>fx</i>	translation force along X axis [N]
<i>fy</i>	translation force along Y axis [N]
<i>fz</i>	translation force along Z axis [N]
<i>t0</i>	wrist joint 0 torque command [Nm]
<i>t1</i>	wrist joint 1 torque command [Nm]
<i>t2</i>	wrist joint 2 torque command [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.135 dhdcSetForceAndWristJointTorquesAndGripperForce()** `int __SDK dhdcSetForceAndWristJointTorquesAndGripperForce (`  
`double fx,`  
`double fy,`  
`double fz,`  
`double t0,`  
`double t1,`  
`double t2,`  
`double fg,`  
`char ID )`

This function sets Cartesian force, wrist joint torques and gripper force.

## Parameters

<i>fx</i>	translation force along X axis [N]
<i>fy</i>	translation force along Y axis [N]
<i>fz</i>	translation force along Z axis [N]
<i>t0</i>	wrist joint 0 torque command [Nm]
<i>t1</i>	wrist joint 1 torque command [Nm]
<i>t2</i>	wrist joint 2 torque command [Nm]
<i>fg</i>	gripper force [N]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.136 dhdcSetGravityCompensation()** `int __SDK dhdcSetGravityCompensation (`  
`int val,`  
`char ID )`

This function toggles the use of the [gravity compensation](#) feature.

## Parameters

<i>val</i>	desired state of the gravity compensation feature (DHD_ON or DHD_OFF)
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.137 dhdSetGripperMotor()** `int __SDK dhdSetGripperMotor (`  
    `ushort mot,`  
    `char ID )`

This function sets a desired motor command to the force gripper.

## Parameters

<i>mot</i>	gripper motor command
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.138 dhdSetMaxForce()** `int __SDK dhdSetMaxForce (`  
    `double f,`  
    `char ID )`

This function defines a limit (in N) to the force magnitude that can be applied by the haptic device. The  $f$  N limit applies to all [dhdSetForce\(\)](#) and related calls, and ensures that the force applied to the device end-effector remains below the requested value. If the  $f$  argument is negative, the limit is disabled and the full range of force available can be applied.



## Parameters

<i>f</i>	the maximum force that can be displayed by the device in [N]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## See also

[dhdGetMaxForce\(\)](#)  
[dhdSetMaxTorque\(\)](#)  
[dhdSetMaxGripperForce\(\)](#)  
[dhdSetMaxPower\(\)](#)  
[dhdSetMaxUsablePower\(\)](#)

## Note

Note that the force limit enforced by [dhdSetMaxForce\(\)](#) only applies to directly will bypass the Cartesian force limit.

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.139 dhdSetMaxGripperForce()** `int __SDK dhdSetMaxGripperForce (`  
    `double f,`  
    `char ID )`

This function defines a limit (in N) to the force magnitude that can be applied by the haptic device gripper. The *f* N limit applies to [dhdSetForceAndTorqueAndGripperForce\(\)](#) and related calls, and ensures that the force applied to the device gripper remains below the requested value. If the *f* argument is negative, the limit is disabled and the full range of gripper force available can be applied.

## Parameters

<i>f</i>	the maximum force that can be displayed by the device gripper in [N]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## See also

[dhdGetMaxGripperForce\(\)](#)  
[dhdSetMaxForce\(\)](#)  
[dhdSetMaxTorque\(\)](#)  
[dhdSetMaxPower\(\)](#)  
[dhdSetMaxUsablePower\(\)](#)

## Note

Note that the force limit enforced by [dhdSetMaxGripperForce\(\)](#) only applies to or motor DAC values directly will bypass the Cartesian force limit.

### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.140 dhdSetMaxTorque()** `int __SDK dhdSetMaxTorque (`  
     `double t,`  
     `char ID )`

This function defines a limit (in Nm) to the torque magnitude that can be applied by the haptic device. The `t` Nm limit applies to all [dhdSetForceAndTorque\(\)](#) and related calls, and ensures that the torque applied to the device end-effector remains below the requested value. If the `t` argument is negative, the limit is disabled and the full range of torque available can be applied.

### Parameters

<i>t</i>	the maximum torque that can be displayed by the device in [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### See also

[dhdGetMaxTorque\(\)](#)  
[dhdSetMaxForce\(\)](#)  
[dhdSetMaxGripperForce\(\)](#)  
[dhdSetMaxPower\(\)](#)  
[dhdSetMaxUsablePower\(\)](#)

### Note

Note that the torque limit enforced by [dhdSetMaxTorque\(\)](#) only applies to DAC values directly will bypass the Cartesian torque limit.

### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.141 dhdSetMot()** `int __SDK dhdSetMot (`  
     `ushort mot[DHD_MAX_DOF],`  
     `uchar mask,`  
     `char ID )`

This function programs motor commands to a selection of motor channels. Particularly useful when using the generic controller directly, without a device model attached.

### Parameters

<i>mot</i>	motor values array
<i>mask</i>	<b>[default=0xff]</b> bitwise mask of which motor should be set
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.142 dhdcSetMotor()** `int __SDK dhdcSetMotor (`  
    `int index,`  
    `ushort val,`  
    `char ID )`

This function programs a command to a single motor channel.

## Parameters

<i>index</i>	the motor index number as defined by <a href="#">DHD_MAX_DOF</a>
<i>val</i>	the motor DAC value
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.143 dhdcSetOutput()** `int __SDK dhdcSetOutput (`  
    `uint output,`  
    `char ID )`

This function sets the user programmable output bits on devices that support it.

## Parameters

<i>output</i>	a bitwise mask that toggles the programmable output bits
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_DELTA3](#)
- [DHD\\_DEVICE\\_SIGMA331](#)

- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.144 dhdSetStandardGravity()** `int __SDK dhdSetStandardGravity (`  
    `double g,`  
    `char ID )`

This function sets the standard gravity constant used in [gravity compensation](#). By default, the constant is set to 9.81 m/s<sup>2</sup>.

#### Parameters

<i>g</i>	standard gravity constant [m/s <sup>2</sup> ]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.145 dhdSetTimeGuard()** `int __SDK dhdSetTimeGuard (`  
    `int us,`  
    `char ID )`

This function toggles the use of the [TimeGuard feature](#) with an arbitrary minimum period.

#### Parameters

<i>us</i>	minimum refresh period in [us] a value of 0 disables the TimeGuard feature, while a value of -1 resets the default value (recommended)
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**[expert mode](#) only - USE AT YOUR OWN RISKS**

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.146 dhdcSetVelocityThreshold()** `int __SDK dhdcSetVelocityThreshold (`  
    `uint val,`  
    `char ID )`

This function adjusts the [velocity threshold](#) of the device. Velocity threshold is a safety feature that prevents the device from accelerating to high velocities without control. If the velocity of one of the device axis passes the threshold, the device enters [BRAKES mode](#).

#### Parameters

<i>val</i>	an arbitrary value of velocity threshold the range of threshold values is device dependent, it is recommended <b>NOT</b> to modify factory settings
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

##### **expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA3](#)
- [DHD\\_DEVICE\\_OMEGA33](#)
- [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)

#### Returns

0 on success, -1 otherwise.

See [error management](#) for details.

**5.1.4.147 dhdcSetVibration()** `int __SDK dhdcSetVibration (`  
    `double freq,`  
    `double amplitude,`  
    `int type,`  
    `char ID )`

This function applies a vibration to the end-effector. The vibration is added to the force requested by [dhdcSetForce\(\)](#) and the like. The vibration application mechanism depends on the specific device type, and is currently only available on devices with dedicated vibration actuators.

#### Parameters

<i>freq</i>	Vibration frequency in Hz
<i>amplitude</i>	Vibration amplitude in N
<i>type</i>	Vibration profile (unused, reserved for future use)
<i>ID</i>	<b>[default=-1]</b> Device ID (see <a href="#">multiple devices</a> section for details)

#### Returns

0 on success, -1 otherwise.

See [error management](#) for details.

**5.1.4.148 dhSetWatchdog()** `int __SDK dhSetWatchdog (`  
    `uchar val,`  
    `char ID )`

This function sets the watchdog duration in multiples of 125 microseconds on compatible devices. If the watchdog duration is exceeded before the device receives a new force command, the device firmware will disable forces. A value of 0 disables the watchdog feature.

#### Parameters

<i>val</i>	watchdog duration in multiples of 125 [us]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

#### See also

[dhGetWatchdog\(\)](#)

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.149 dhSetWristJointTorques()** `int __SDK dhSetWristJointTorques (`  
    `double t0,`  
    `double t1,`  
    `double t2,`  
    `char ID )`

This function sets all joint torques of the wrist structure.

#### Parameters

<i>t0</i>	wrist joint 0 torque command [Nm]
<i>t1</i>	wrist joint 1 torque command [Nm]
<i>t2</i>	wrist joint 2 torque command [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

#### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.150 dhdcSetWristMotor()** `int __SDK dhdcSetWristMotor (`  
    `ushort mot0,`  
    `ushort mot1,`  
    `ushort mot2,`  
    `char ID )`

This function sets desired motor commands to the amplifier channels commanding the wrist motors.

#### Parameters

<i>mot0</i>	wrist joint 0 motor command
<i>mot1</i>	wrist joint 1 motor command
<i>mot2</i>	wrist joint 2 motor command
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)

#### Returns

0 on success, -1 otherwise.

See [error management](#) for details.

**5.1.4.151 dhdcSleep()** `void __SDK dhdcSleep (`  
    `double sec )`

This function sleeps for a given period of time (OS independent).

#### Parameters

<i>sec</i>	sleep period in [s]
------------	---------------------

**5.1.4.152 dhdcStartThread()** `int __SDK dhdcStartThread (`  
    `void * funcvoid *,`  
    `void * arg,`  
    `int priority )`

This function creates a thread on any operating systems supported by the SDK.

#### Note

This is a convenience function designed for simple, portable, multi-threaded applications. It is not intended to replace native OS functions. Force Dimension recommends using the native OS thread libraries in any application that requires reliable parallel computing.

**Parameters**

<i>func</i>	function to run in the thread
<i>arg</i>	optional pointer to an argument passed to the thread function
<i>priority</i>	priority given to the thread (see the <a href="#">multi-threading</a> section for details). The SDK will try to set the priority level, but will continue without error if the OS does not accept the request.

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.153 dhdStop()** `int __SDK dhdStop (`  
`char ID )`

This function stops all forces on the device. This routine disables the force on the haptic device and puts it into BRAKE [mode](#).

**Parameters**

<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.154 dhdUpdateEncoders()** `int __SDK dhdUpdateEncoders (`  
`char ID )`

This function forces an update of the internal encoder values in the state vector. This call retrieves the encoder readings from the device and places them into the state vector. No kinematic model is called.

**Parameters**

<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)
-----------	--

**Note**

**[expert mode](#) only - USE AT YOUR OWN RISKS**

**Returns**

0 on success, -1 otherwise.  
See [error management](#) for details.



**5.1.4.155 dhdWaitForReset()** `int __SDK dhdWaitForReset (`  
     `int timeout,`  
     `char ID )`

This function puts the device in [RESET mode](#) and wait for the user to [calibrate](#) the device. Optionally, a timeout can be defined after which the call returns even if calibration has not occurred.

#### Parameters

<i>timeout</i>	[optional] maximum time to wait for calibration in [ms]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

If the timeout is reached, the call returns an error (-1) and [dhdErrno](#) is set to DHD\_ERROR\_TIMEOUT.

#### Returns

0 on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.156 dhdWristEncodersToJointAngles()** `int __SDK dhdWristEncodersToJointAngles (`  
     `int enc0,`  
     `int enc1,`  
     `int enc2,`  
     `double * j0,`  
     `double * j1,`  
     `double * j2,`  
     `char ID )`

This function computes and returns the wrist joint angles for a given set of encoder readings.

#### Parameters

<i>enc0</i>	wrist encoder reading on axis 0
<i>enc1</i>	wrist encoder reading on axis 1
<i>enc2</i>	wrist encoder reading on axis 2
<i>j0</i>	<b>[out]</b> joint angle for wrist axis 0 in [rad]
<i>j1</i>	<b>[out]</b> joint angle for wrist axis 1 in [rad]
<i>j2</i>	<b>[out]</b> joint angle for wrist axis 2 in [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**[expert mode](#) only - USE AT YOUR OWN RISKS**

#### Returns

0 on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.157 dhdWristEncoderToOrientation()** `int __SDK dhdWristEncoderToOrientation (`  
`int enc0,`  
`int enc1,`  
`int enc2,`  
`double * oa,`  
`double * ob,`  
`double * og,`  
`char ID )`

For devices with a wrist structure, this function computes individual angle of each joint, starting with the one located nearest to the wrist base plate. For the [DHD\\_DEVICE\\_OMEGA33](#) and [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#) devices, angles are computed with respect to their internal reference frame, which is rotated 45 degrees around the Y axis. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>enc0</i>	wrist encoder reading on axis 0
<i>enc1</i>	wrist encoder reading on axis 1
<i>enc2</i>	wrist encoder reading on axis 2
<i>oa</i>	<b>[out]</b> wrist end-effector orientation around the first wrist joint [rad]
<i>ob</i>	<b>[out]</b> wrist end-effector orientation around the second wrist joint [rad]
<i>og</i>	<b>[out]</b> wrist end-effector orientation around the third wrist joint [rad]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

##### **expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA33](#)
- [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

#### Returns

0 on success, -1 otherwise.

See [error management](#) for details.

**5.1.4.158 dhdWristGravityJointTorques()** `int __SDK dhdWristGravityJointTorques (`  
`double j0,`  
`double j1,`  
`double j2,`  
`double * q0,`  
`double * q1,`  
`double * q2,`  
`char ID )`

This function computes the wrist joint torques required to compensate for gravity in a given wrist joint angle configuration. Please refer to your device user manual for more information on your device coordinate system.

## Parameters

<i>j0</i>	joint angle for wrist axis 0 in [rad]
<i>j1</i>	joint angle for wrist axis 1 in [rad]
<i>j2</i>	joint angle for wrist axis 2 in [rad]
<i>q0</i>	<b>out</b> gravity compensation joint torque on axis 0 in [Nm]
<i>q1</i>	<b>out</b> gravity compensation joint torque on axis 1 in [Nm]
<i>q2</i>	<b>out</b> gravity compensation joint torque on axis 2 in [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

```

5.1.4.159 dhcWristJointAnglesToEncoders() int __SDK dhcWristJointAnglesToEncoders (
    double j0,
    double j1,
    double j2,
    int * enc0,
    int * enc1,
    int * enc2,
    char ID )

```

This function computes and returns the wrist encoder readings for a given set of joint angles.

## Parameters

<i>j0</i>	joint angle for axis 0 in [rad]
<i>j1</i>	joint angle for axis 1 in [rad]
<i>j2</i>	joint angle for axis 2 in [rad]
<i>enc0</i>	<b>[out]</b> wrist encoder reading on axis 0
<i>enc1</i>	<b>[out]</b> wrist encoder reading on axis 1
<i>enc2</i>	<b>[out]</b> wrist encoder reading on axis 2
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

## Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.160 dhdWristJointAnglesToJacobian()** `int __SDK dhdWristJointAnglesToJacobian (`  
`double j0,`  
`double j1,`  
`double j2,`  
`double jcb[3][3],`  
`char ID )`

This function retrieves the wrist jacobian matrix based on a given joint configuration. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>j0</i>	joint angle for wrist axis 0 in [rad]
<i>j1</i>	joint angle for wrist axis 1 in [rad]
<i>j2</i>	joint angle for wrist axis 2 in [rad]
<i>jcb</i>	<b>[out]</b> device jacobian
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

#### Returns

0 on success, -1 otherwise.  
 See [error management](#) for details.

**5.1.4.161 dhdWristJointTorquesExtrema()** `int __SDK dhdWristJointTorquesExtrema (`  
`double j0,`  
`double j1,`  
`double j2,`  
`double minq[3],`  
`double maxq[3],`  
`char ID )`

This function computes the range of applicable wrist joint torques for a given wrist joint angle configuration. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>j0</i>	joint angle for wrist axis 0 in [rad]
<i>j1</i>	joint angle for wrist axis 1 in [rad]
<i>j2</i>	joint angle for wrist axis 2 in [rad]
<i>minq</i>	<b>outarray</b> of minimum applicable joint torque in [Nm]
<i>maxq</i>	<b>outarray</b> of maximum applicable joint torque in [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

**expert mode only - USE AT YOUR OWN RISKS**

### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.162 dhdWristMotorToTorque()** `int __SDK dhdWristMotorToTorque (`  
    `ushort mot0,`  
    `ushort mot1,`  
    `ushort mot2,`  
    `int enc0,`  
    `int enc1,`  
    `int enc2,`  
    `double * tx,`  
    `double * ty,`  
    `double * tz,`  
    `char ID )`

This function computes and returns the torque applied to the end-effector for a given set of motor commands at a given orientation (defined by encoder readings).

### Parameters

<i>mot0</i>	motor command around wrist joint 0
<i>mot1</i>	motor command around wrist joint 1
<i>mot2</i>	motor command around wrist joint 2
<i>enc0</i>	wrist encoder reading around axis 0
<i>enc1</i>	wrist encoder reading around axis 1
<i>enc2</i>	wrist encoder reading around axis 2
<i>tx</i>	<b>[out]</b> torque on the wrist end-effector around the X axis [Nm]
<i>ty</i>	<b>[out]</b> torque on the wrist end-effector around the Y axis [Nm]
<i>tz</i>	<b>[out]</b> torque on the wrist end-effector around the Z axis [Nm]
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

### Note

**expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)

### Returns

0 on success, -1 otherwise.  
See [error management](#) for details.

**5.1.4.163 dhdWristOrientationToEncoder()** `int __SDK dhdWristOrientationToEncoder (`  
`double oa,`  
`double ob,`  
`double og,`  
`int * enc0,`  
`int * enc1,`  
`int * enc2,`  
`char ID )`

For devices with a wrist structure, this function computes the encoder values from individual angle of each joint, starting with the one located nearest to the wrist base plate. For the [DHD\\_DEVICE\\_OMEGA33](#) and [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#) devices, angles must be expressed with respect to their internal reference frame, which is rotated 45 degrees around the Y axis. Please refer to your device user manual for more information on your device coordinate system.

#### Parameters

<i>oa</i>	wrist end-effector orientation around the X axis [rad]
<i>ob</i>	wrist end-effector orientation around the Y axis [rad]
<i>og</i>	wrist end-effector orientation around the Z axis [rad]
<i>enc0</i>	<b>[out]</b> wrist encoder reading on first joint
<i>enc1</i>	<b>[out]</b> wrist encoder reading on second joint
<i>enc2</i>	<b>[out]</b> wrist encoder reading on third joint
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

#### Note

##### **expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_OMEGA33](#)
- [DHD\\_DEVICE\\_OMEGA33\\_LEFT](#)
- [DHD\\_DEVICE\\_OMEGA331](#)
- [DHD\\_DEVICE\\_OMEGA331\\_LEFT](#)
- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)
- [DHD\\_DEVICE\\_LAMBDA331](#)
- [DHD\\_DEVICE\\_LAMBDA331\\_LEFT](#)

#### Returns

0 on success, -1 otherwise.

See [error management](#) for details.

**5.1.4.164 dhdWristTorqueToMotor()** `int __SDK dhdWristTorqueToMotor (`  
`double tx,`  
`double ty,`  
`double tz,`  
`int enc0,`  
`int enc1,`  
`int enc2,`  
`ushort * mot0,`  
`ushort * mot1,`  
`ushort * mot2,`  
`char ID )`

This function computes and returns the motor command necessary to obtain a given torque on the end-effector at a given orientation (defined by encoder readings).

## Parameters

<i>tx</i>	torque on the wrist end-effector around the X axis [Nm]
<i>ty</i>	torque on the wrist end-effector around the Y axis [Nm]
<i>tz</i>	torque on the wrist end-effector around the Z axis [Nm]
<i>enc0</i>	wrist encoder reading on axis 0
<i>enc1</i>	wrist encoder reading on axis 1
<i>enc2</i>	wrist encoder reading on axis 2
<i>mot0</i>	<b>[out]</b> motor command around wrist joint 0
<i>mot1</i>	<b>[out]</b> motor command around wrist joint 1
<i>mot2</i>	<b>[out]</b> motor command around wrist joint 2
<i>ID</i>	<b>[default=-1]</b> device ID (see <a href="#">multiple devices</a> section for details)

## Note

**expert mode only - USE AT YOUR OWN RISKS**

This feature only applies to the following devices:

- [DHD\\_DEVICE\\_SIGMA331](#)
- [DHD\\_DEVICE\\_SIGMA331\\_LEFT](#)

## Returns

0 or [DHD\\_MOTOR\\_SATURATED](#) on success, -1 otherwise.

See [error management](#) for details.

## 6 Example Documentation

### 6.1 hello\_world.cpp

The "hello world" DHD application.

```

////////////////////////////////////
//
// This example implements a simple spring model which pulls the device
// towards the center of the workspace. If the user presses the user button,
// the application exits.
//
////////////////////////////////////

// C++ library headers
#include <iostream>
#include <iomanip>
// project headers
#include "dhdc.h"
// constants
constexpr double K = 1000.0;

////////////////////////////////////

int main(int argc,
         char* argv[])
{
    // open a connection to the device
    if (dhdOpen() < 0)
    {
        std::cout << "error: cannot open device" << std::endl;
        return -1;
    }
    // run haptic loop
    int done = 0;
    while (!done)
    {
        // get end-effector position
        double px, py, pz;
        dhdGetPosition(&px, &py, &pz);
    }
}

```

```

        // compute spring model
        double fx, fy, fz;
        fx = -K * px;
        fy = -K * py;
        fz = -K * pz;
        // apply forces
        dhdcSetForce(fx, fy, fz);
        // exit if the button is pushed
        done += dhdcGetButton(0);
    }
    // close the connection
    dhdcClose();
    return 0;
}

```

## 6.2 multiple\_devices.cpp

Simple example of multiple-devices programming.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// This example implements a simple gravity compensation loop for two devices.
// Note that a more efficient approach would require the creation of two haptic
// threads, one for each device. This implementation is OS specific however
// and is not implemented here.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// C++ library headers
#include <iostream>
#include <iomanip>
// project headers
#include "dhdc.h"

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main(int argc,
         char* argv[])
{
    // get device count
    int deviceCount = dhdcGetDeviceCount();
    if (deviceCount < 0)
    {
        std::cout << "error: " << dhdcErrorGetLastStr() << std::endl;
        return -1;
    }
    else if (deviceCount < 1)
    {
        std::cout << "error: no device detected" << std::endl;
    }
    else if (deviceCount < 2)
    {
        std::cout << "error: singled device detected" << std::endl;
    }
    // open the first available device
    int deviceID0 = dhdcOpenID(0);
    if (deviceID0 < 0)
    {
        std::cout << "error: " << dhdcErrorGetLastStr() << std::endl;
        return -1;
    }
    // open the second available device
    int deviceID1 = dhdcOpenID(1);
    if (deviceID1 < 0)
    {
        std::cout << "error: " << dhdcErrorGetLastStr() << std::endl;
        return -1;
    }
    // haptic loop
    while (true)
    {
        // apply a null force to put the first device in gravity compensation
        if (dhdcSetForce(0.0, 0.0, 0.0, deviceID0) < 0)
        {
            std::cout << "error: " << dhdcErrorGetLastStr() << std::endl;
            break;
        }
        // apply a null force to put the second device in gravity compensation
        if (dhdcSetForce(0.0, 0.0, 0.0, deviceID1) < 0)
        {
            std::cout << "error: " << dhdcErrorGetLastStr() << std::endl;
            break;
        }
        // exit the haptic loop on button press
        if (dhdcGetButton(0, deviceID0) || dhdcGetButton(0, deviceID1))
        {
            break;
        }
    }
}

```



```

    }
}
// close the connection to the first device
if (dhdClose(deviceID0) < 0)
{
    std::cout << "error: " << dhdErrorGetLastStr() << std::endl;
}
// close the connection to the second device
if (dhdClose(deviceID1) < 0)
{
    std::cout << "error: " << dhdErrorGetLastStr() << std::endl;
}
return 0;
}

```

## 6.3 single\_device.cpp

Simple example of single-device programming.

```

////////////////////////////////////
//
// This example implements a simple gravity compensation loop for a single
// haptic device.
//
////////////////////////////////////

// C++ library headers
#include <iostream>
#include <iomanip>
// project headers
#include "dhdc.h"

////////////////////////////////////

int main(int argc,
         char* argv[])
{
    // get device count
    if (dhdGetDeviceCount() <= 0)
    {
        std::cout << "error: " << dhdErrorGetLastStr() << std::endl;
        return -1;
    }
    // open the first available device
    if (dhdOpen() < 0)
    {
        std::cout << "error: " << dhdErrorGetLastStr() << std::endl;
        return -1;
    }
    // haptic loop
    while (true)
    {
        // apply a null force to put the device in gravity compensation
        if (dhdSetForce(0.0, 0.0, 0.0) < 0)
        {
            std::cout << "error: " << dhdErrorGetLastStr() << std::endl;
            break;
        }
        // exit the haptic loop on button press
        if (dhdGetButton(0))
        {
            break;
        }
    }
    // close the connection to the device
    if (dhdClose() < 0)
    {
        std::cout << "error: " << dhdErrorGetLastStr() << std::endl;
    }
    return 0;
}

```



## Index

DHD\_COM\_MODE\_ASYNC  
dhdc.h, [17](#)

DHD\_COM\_MODE\_NETWORK  
dhdc.h, [17](#)

DHD\_COM\_MODE\_SYNC  
dhdc.h, [17](#)

DHD\_COM\_MODE\_VIRTUAL  
dhdc.h, [17](#)

DHD\_DELTA\_ENC\_0  
dhdc.h, [17](#)

DHD\_DELTA\_ENC\_1  
dhdc.h, [18](#)

DHD\_DELTA\_ENC\_2  
dhdc.h, [18](#)

DHD\_DELTA\_MOTOR\_0  
dhdc.h, [18](#)

DHD\_DELTA\_MOTOR\_1  
dhdc.h, [18](#)

DHD\_DELTA\_MOTOR\_2  
dhdc.h, [18](#)

DHD\_DEVICE\_CONTROLLER  
dhdc.h, [18](#)

DHD\_DEVICE\_CONTROLLER\_HR  
dhdc.h, [18](#)

DHD\_DEVICE\_CUSTOM  
dhdc.h, [18](#)

DHD\_DEVICE\_DELTA3  
dhdc.h, [18](#)

DHD\_DEVICE\_FALCON  
dhdc.h, [18](#)

DHD\_DEVICE\_LAMBDA331  
dhdc.h, [18](#)

DHD\_DEVICE\_LAMBDA331\_LEFT  
dhdc.h, [18](#)

DHD\_DEVICE\_NONE  
dhdc.h, [19](#)

DHD\_DEVICE\_OMEGA3  
dhdc.h, [19](#)

DHD\_DEVICE\_OMEGA33  
dhdc.h, [19](#)

DHD\_DEVICE\_OMEGA331  
dhdc.h, [19](#)

DHD\_DEVICE\_OMEGA331\_LEFT  
dhdc.h, [19](#)

DHD\_DEVICE\_OMEGA33\_LEFT  
dhdc.h, [19](#)

DHD\_DEVICE\_SIGMA331  
dhdc.h, [19](#)

DHD\_DEVICE\_SIGMA331\_LEFT  
dhdc.h, [19](#)

DHD\_ERROR  
dhdc.h, [24](#)

DHD\_ERROR\_COM  
dhdc.h, [24](#)

DHD\_ERROR\_CONFIGURATION  
dhdc.h, [24](#)

DHD\_ERROR\_DEPRECATED  
dhdc.h, [24](#)

DHD\_ERROR\_DEVICE\_IN\_USE  
dhdc.h, [24](#)

DHD\_ERROR\_DEVICE\_NOT\_READY  
dhdc.h, [24](#)

DHD\_ERROR\_DHC\_BUSY  
dhdc.h, [24](#)

DHD\_ERROR\_EXPERT\_MODE\_DISABLED  
dhdc.h, [24](#)

DHD\_ERROR\_FILE\_NOT\_FOUND  
dhdc.h, [24](#)

DHD\_ERROR\_GEOMETRY  
dhdc.h, [24](#)

DHD\_ERROR\_INVALID  
dhdc.h, [24](#)

DHD\_ERROR\_INVALID\_INDEX  
dhdc.h, [24](#)

DHD\_ERROR\_NO\_DEVICE\_FOUND  
dhdc.h, [24](#)

DHD\_ERROR\_NO\_DRIVER\_FOUND  
dhdc.h, [24](#)

DHD\_ERROR\_NO\_REGULATION  
dhdc.h, [24](#)

DHD\_ERROR\_NOT\_AVAILABLE  
dhdc.h, [24](#)

DHD\_ERROR\_NOT\_ENABLED  
dhdc.h, [24](#)

DHD\_ERROR\_NOT\_IMPLEMENTED  
dhdc.h, [24](#)

DHD\_ERROR\_NULL\_ARGUMENT  
dhdc.h, [24](#)

DHD\_ERROR\_OUT\_OF\_MEMORY  
dhdc.h, [24](#)

DHD\_ERROR\_REDUNDANT\_FAIL  
dhdc.h, [24](#)

DHD\_ERROR\_TIMEOUT  
dhdc.h, [24](#)

dhd\_errors  
dhdc.h, [23](#)

DHD\_MAX\_BUTTONS  
dhdc.h, [19](#)

DHD\_MAX\_DOF  
dhdc.h, [19](#)

DHD\_MAX\_STATUS  
dhdc.h, [19](#)

DHD\_MOTOR\_SATURATED  
dhdc.h, [19](#)

DHD\_NO\_ERROR  
dhdc.h, [24](#)

DHD\_OFF  
dhdc.h, [20](#)

DHD\_ON  
dhdc.h, [20](#)

DHD\_STATUS\_AXIS\_CHECKED  
dhdc.h, [20](#)

DHD\_STATUS\_BRAKE  
dhdc.h, [20](#)

DHD\_STATUS\_CONNECTED  
dhdc.h, [20](#)

DHD\_STATUS\_ERROR  
dhdc.h, 20  
DHD\_STATUS\_FORCE  
dhdc.h, 20  
DHD\_STATUS\_FORCE\_OFF\_CAUSE  
dhdc.h, 20  
DHD\_STATUS\_GRAVITY  
dhdc.h, 21  
DHD\_STATUS\_IDLE  
dhdc.h, 21  
DHD\_STATUS\_LOCKS  
dhdc.h, 21  
DHD\_STATUS\_POWER  
dhdc.h, 21  
DHD\_STATUS\_REDUNDANCY  
dhdc.h, 21  
DHD\_STATUS\_RESET  
dhdc.h, 21  
DHD\_STATUS\_STARTED  
dhdc.h, 21  
DHD\_STATUS\_TIMEGUARD  
dhdc.h, 21  
DHD\_STATUS\_TORQUE  
dhdc.h, 22  
DHD\_STATUS\_WRIST\_DETECTED  
dhdc.h, 22  
DHD\_STATUS\_WRIST\_INIT  
dhdc.h, 22  
DHD\_THREAD\_PRIORITY\_DEFAULT  
dhdc.h, 22  
DHD\_THREAD\_PRIORITY\_HIGH  
dhdc.h, 22  
DHD\_THREAD\_PRIORITY\_LOW  
dhdc.h, 22  
DHD\_TIMEGUARD  
dhdc.h, 22  
DHD\_UNDEFINED  
dhdc.h, 22  
DHD\_VELOCITY\_WINDOW  
dhdc.h, 22  
DHD\_VELOCITY\_WINDOWING  
dhdc.h, 22  
DHD\_WRIST\_ENC\_0  
dhdc.h, 23  
DHD\_WRIST\_ENC\_1  
dhdc.h, 23  
DHD\_WRIST\_ENC\_2  
dhdc.h, 23  
DHD\_WRIST\_MOTOR\_0  
dhdc.h, 23  
DHD\_WRIST\_MOTOR\_1  
dhdc.h, 23  
DHD\_WRIST\_MOTOR\_2  
dhdc.h, 23  
dhdc.h, 12  
DHD\_COM\_MODE\_ASYNC, 17  
DHD\_COM\_MODE\_NETWORK, 17  
DHD\_COM\_MODE\_SYNC, 17  
DHD\_COM\_MODE\_VIRTUAL, 17  
DHD\_DELTA\_ENC\_0, 17  
DHD\_DELTA\_ENC\_1, 18  
DHD\_DELTA\_ENC\_2, 18  
DHD\_DELTA\_MOTOR\_0, 18  
DHD\_DELTA\_MOTOR\_1, 18  
DHD\_DELTA\_MOTOR\_2, 18  
DHD\_DEVICE\_CONTROLLER, 18  
DHD\_DEVICE\_CONTROLLER\_HR, 18  
DHD\_DEVICE\_CUSTOM, 18  
DHD\_DEVICE\_DELTA3, 18  
DHD\_DEVICE\_FALCON, 18  
DHD\_DEVICE\_LAMBDA331, 18  
DHD\_DEVICE\_LAMBDA331\_LEFT, 18  
DHD\_DEVICE\_NONE, 19  
DHD\_DEVICE\_OMEGA3, 19  
DHD\_DEVICE\_OMEGA33, 19  
DHD\_DEVICE\_OMEGA331, 19  
DHD\_DEVICE\_OMEGA331\_LEFT, 19  
DHD\_DEVICE\_OMEGA33\_LEFT, 19  
DHD\_DEVICE\_SIGMA331, 19  
DHD\_DEVICE\_SIGMA331\_LEFT, 19  
DHD\_ERROR, 24  
DHD\_ERROR\_COM, 24  
DHD\_ERROR\_CONFIGURATION, 24  
DHD\_ERROR\_DEPRECATED, 24  
DHD\_ERROR\_DEVICE\_IN\_USE, 24  
DHD\_ERROR\_DEVICE\_NOT\_READY, 24  
DHD\_ERROR\_DHC\_BUSY, 24  
DHD\_ERROR\_EXPERT\_MODE\_DISABLED, 24  
DHD\_ERROR\_FILE\_NOT\_FOUND, 24  
DHD\_ERROR\_GEOMETRY, 24  
DHD\_ERROR\_INVALID, 24  
DHD\_ERROR\_INVALID\_INDEX, 24  
DHD\_ERROR\_NO\_DEVICE\_FOUND, 24  
DHD\_ERROR\_NO\_DRIVER\_FOUND, 24  
DHD\_ERROR\_NO\_REGULATION, 24  
DHD\_ERROR\_NOT\_AVAILABLE, 24  
DHD\_ERROR\_NOT\_ENABLED, 24  
DHD\_ERROR\_NOT\_IMPLEMENTED, 24  
DHD\_ERROR\_NULL\_ARGUMENT, 24  
DHD\_ERROR\_OUT\_OF\_MEMORY, 24  
DHD\_ERROR\_REDUNDANT\_FAIL, 24  
DHD\_ERROR\_TIMEOUT, 24  
dhd\_errors, 23  
DHD\_MAX\_BUTTONS, 19  
DHD\_MAX\_DOF, 19  
DHD\_MAX\_STATUS, 19  
DHD\_MOTOR SATURATED, 19  
DHD\_NO\_ERROR, 24  
DHD\_OFF, 20  
DHD\_ON, 20  
DHD\_STATUS\_AXIS\_CHECKED, 20  
DHD\_STATUS\_BRAKE, 20  
DHD\_STATUS\_CONNECTED, 20  
DHD\_STATUS\_ERROR, 20  
DHD\_STATUS\_FORCE, 20  
DHD\_STATUS\_FORCE\_OFF\_CAUSE, 20  
DHD\_STATUS\_GRAVITY, 21  
DHD\_STATUS\_IDLE, 21  
DHD\_STATUS\_LOCKS, 21  
DHD\_STATUS\_POWER, 21  
DHD\_STATUS\_REDUNDANCY, 21  
DHD\_STATUS\_RESET, 21

DHD\_STATUS\_STARTED, [21](#)  
DHD\_STATUS\_TIMEGUARD, [21](#)  
DHD\_STATUS\_TORQUE, [22](#)  
DHD\_STATUS\_WRIST\_DETECTED, [22](#)  
DHD\_STATUS\_WRIST\_INIT, [22](#)  
DHD\_THREAD\_PRIORITY\_DEFAULT, [22](#)  
DHD\_THREAD\_PRIORITY\_HIGH, [22](#)  
DHD\_THREAD\_PRIORITY\_LOW, [22](#)  
DHD\_TIMEGUARD, [22](#)  
DHD\_UNDEFINED, [22](#)  
DHD\_VELOCITY\_WINDOW, [22](#)  
DHD\_VELOCITY\_WINDOWING, [22](#)  
DHD\_WRIST\_ENC\_0, [23](#)  
DHD\_WRIST\_ENC\_1, [23](#)  
DHD\_WRIST\_ENC\_2, [23](#)  
DHD\_WRIST\_MOTOR\_0, [23](#)  
DHD\_WRIST\_MOTOR\_1, [23](#)  
DHD\_WRIST\_MOTOR\_2, [23](#)  
dhdCalibrateWrist, [24](#)  
dhdCheckControllerMemory, [25](#)  
dhdClose, [25](#)  
dhdConfigAngularVelocity, [26](#)  
dhdConfigGripperVelocity, [26](#)  
dhdConfigLinearVelocity, [27](#)  
dhdControllerSetDevice, [27](#)  
dhdDeltaEncodersToJointAngles, [28](#)  
dhdDeltaEncoderToPosition, [28](#)  
dhdDeltaForceToMotor, [29](#)  
dhdDeltaGravityJointTorques, [30](#)  
dhdDeltaJointAnglesToEncoders, [30](#)  
dhdDeltaJointAnglesToJacobian, [31](#)  
dhdDeltaJointTorquesExtrema, [31](#)  
dhdDeltaMotorToForce, [32](#)  
dhdDeltaPositionToEncoder, [33](#)  
dhdDisableExpertMode, [33](#)  
dhdEmulateButton, [33](#)  
dhdEnableExpertMode, [34](#)  
dhdEnableForce, [34](#)  
dhdEnableGripperForce, [35](#)  
dhdEnableSimulator, [35](#)  
dhdErrorGetLast, [35](#)  
dhdErrorGetLastStr, [36](#)  
dhdErrorGetStr, [36](#)  
dhdGetAngularVelocityDeg, [36](#)  
dhdGetAngularVelocityRad, [37](#)  
dhdGetAvailableCount, [38](#)  
dhdGetBaseAngleXDeg, [38](#)  
dhdGetBaseAngleXRad, [39](#)  
dhdGetBaseAngleZDeg, [39](#)  
dhdGetBaseAngleZRad, [39](#)  
dhdGetButton, [41](#)  
dhdGetButtonMask, [41](#)  
dhdGetComFreq, [42](#)  
dhdGetComMode, [42](#)  
dhdGetComponentVersionStr, [42](#)  
dhdGetDeltaEncoders, [43](#)  
dhdGetDeltaJacobian, [43](#)  
dhdGetDeltaJointAngles, [43](#)  
dhdGetDeviceAngleDeg, [44](#)  
dhdGetDeviceAngleRad, [44](#)  
dhdGetDeviceCount, [45](#)  
dhdGetDeviceID, [45](#)  
dhdGetEffectorMass, [45](#)  
dhdGetEnc, [46](#)  
dhdGetEncoder, [46](#)  
dhdGetEncRange, [47](#)  
dhdGetEncVelocities, [47](#)  
dhdGetForce, [48](#)  
dhdGetForceAndTorque, [48](#)  
dhdGetForceAndTorqueAndGripperForce, [49](#)  
dhdGetGripperAngleDeg, [49](#)  
dhdGetGripperAngleRad, [50](#)  
dhdGetGripperAngularVelocityDeg, [50](#)  
dhdGetGripperAngularVelocityRad, [51](#)  
dhdGetGripperEncoder, [52](#)  
dhdGetGripperFingerPos, [52](#)  
dhdGetGripperGap, [53](#)  
dhdGetGripperLinearVelocity, [53](#)  
dhdGetGripperThumbPos, [54](#)  
dhdGetJointAngleRange, [55](#)  
dhdGetJointAngles, [55](#)  
dhdGetJointVelocities, [55](#)  
dhdGetLinearVelocity, [56](#)  
dhdGetMaxForce, [56](#)  
dhdGetMaxGripperForce, [57](#)  
dhdGetMaxTorque, [57](#)  
dhdGetOrientationDeg, [58](#)  
dhdGetOrientationFrame, [59](#)  
dhdGetOrientationRad, [59](#)  
dhdGetPosition, [60](#)  
dhdGetPositionAndOrientationDeg, [60](#)  
dhdGetPositionAndOrientationFrame, [61](#)  
dhdGetPositionAndOrientationRad, [61](#)  
dhdGetSDKVersion, [62](#)  
dhdGetSDKVersionStr, [63](#)  
dhdGetSerialNumber, [63](#)  
dhdGetStatus, [63](#)  
dhdGetSystemCounter, [63](#)  
dhdGetSystemName, [64](#)  
dhdGetSystemRev, [64](#)  
dhdGetSystemType, [64](#)  
dhdGetTime, [65](#)  
dhdGetVelocityThreshold, [65](#)  
dhdGetVersion, [66](#)  
dhdGetVersionStr, [66](#)  
dhdGetWatchdog, [66](#)  
dhdGetWristEncoders, [67](#)  
dhdGetWristJacobian, [68](#)  
dhdGetWristJointAngles, [68](#)  
dhdGripperAngleRadToEncoder, [68](#)  
dhdGripperEncoderToAngleRad, [69](#)  
dhdGripperEncoderToGap, [70](#)  
dhdGripperForceToMotor, [70](#)  
dhdGripperGapToEncoder, [71](#)  
dhdGripperMotorToForce, [72](#)  
dhdHasActiveGripper, [72](#)  
dhdHasActiveWrist, [73](#)  
dhdHasBase, [73](#)  
dhdHasGripper, [74](#)  
dhdHasWrist, [74](#)  
dhdIsLeftHanded, [75](#)  
dhdJointAnglesToInertiaMatrix, [76](#)

dhdKbGet, 76  
 dhdKbHit, 76  
 dhdOpen, 76  
 dhdOpenID, 77  
 dhdOpenSerial, 77  
 dhdOpenType, 78  
 dhdPreloadMot, 78  
 dhdPreset, 79  
 dhdReadConfigFromFile, 79  
 dhdReset, 81  
 dhdResetWrist, 81  
 dhdSetBaseAngleXDeg, 81  
 dhdSetBaseAngleXRad, 82  
 dhdSetBaseAngleZDeg, 82  
 dhdSetBaseAngleZRad, 83  
 dhdSetBrakes, 83  
 dhdSetBrk, 84  
 dhdSetComMode, 84  
 dhdSetComModePriority, 85  
 dhdSetDeltaJointTorques, 85  
 dhdSetDeltaMotor, 86  
 dhdSetDevice, 86  
 dhdSetDeviceAngleDeg, 87  
 dhdSetDeviceAngleRad, 87  
 dhdSetEffectorMass, 88  
 dhdSetForce, 88  
 dhdSetForceAndGripperForce, 88  
 dhdSetForceAndTorque, 89  
 dhdSetForceAndTorqueAndGripperForce, 89  
 dhdSetForceAndWristJointTorques, 90  
 dhdSetForceAndWristJointTorquesAndGripperForce, 91  
 dhdSetGravityCompensation, 91  
 dhdSetGripperMotor, 92  
 dhdSetMaxForce, 92  
 dhdSetMaxGripperForce, 93  
 dhdSetMaxTorque, 94  
 dhdSetMot, 94  
 dhdSetMotor, 95  
 dhdSetOutput, 95  
 dhdSetStandardGravity, 96  
 dhdSetTimeGuard, 96  
 dhdSetVelocityThreshold, 96  
 dhdSetVibration, 97  
 dhdSetWatchdog, 97  
 dhdSetWristJointTorques, 98  
 dhdSetWristMotor, 98  
 dhdSleep, 99  
 dhdStartThread, 99  
 dhdStop, 100  
 dhdUpdateEncoders, 100  
 dhdWaitForReset, 100  
 dhdWristEncodersToJointAngles, 101  
 dhdWristEncoderToOrientation, 101  
 dhdWristGravityJointTorques, 102  
 dhdWristJointAnglesToEncoders, 103  
 dhdWristJointAnglesToJacobian, 103  
 dhdWristJointTorquesExtrema, 104  
 dhdWristMotorToTorque, 105  
 dhdWristOrientationToEncoder, 105  
 dhdWristTorqueToMotor, 106  
 dhdCalibrateWrist  
     dhdc.h, 24  
 dhdCheckControllerMemory  
     dhdc.h, 25  
 dhdClose  
     dhdc.h, 25  
 dhdConfigAngularVelocity  
     dhdc.h, 26  
 dhdConfigGripperVelocity  
     dhdc.h, 26  
 dhdConfigLinearVelocity  
     dhdc.h, 27  
 dhdControllerSetDevice  
     dhdc.h, 27  
 dhdDeltaEncodersToJointAngles  
     dhdc.h, 28  
 dhdDeltaEncoderToPosition  
     dhdc.h, 28  
 dhdDeltaForceToMotor  
     dhdc.h, 29  
 dhdDeltaGravityJointTorques  
     dhdc.h, 30  
 dhdDeltaJointAnglesToEncoders  
     dhdc.h, 30  
 dhdDeltaJointAnglesToJacobian  
     dhdc.h, 31  
 dhdDeltaJointTorquesExtrema  
     dhdc.h, 31  
 dhdDeltaMotorToForce  
     dhdc.h, 32  
 dhdDeltaPositionToEncoder  
     dhdc.h, 33  
 dhdDisableExpertMode  
     dhdc.h, 33  
 dhdEmulateButton  
     dhdc.h, 33  
 dhdEnableExpertMode  
     dhdc.h, 34  
 dhdEnableForce  
     dhdc.h, 34  
 dhdEnableGripperForce  
     dhdc.h, 35  
 dhdEnableSimulator  
     dhdc.h, 35  
 dhdErrorGetLast  
     dhdc.h, 35  
 dhdErrorGetLastStr  
     dhdc.h, 36  
 dhdErrorGetStr  
     dhdc.h, 36  
 dhdGetAngularVelocityDeg  
     dhdc.h, 36  
 dhdGetAngularVelocityRad  
     dhdc.h, 37  
 dhdGetAvailableCount  
     dhdc.h, 38  
 dhdGetBaseAngleXDeg  
     dhdc.h, 38  
 dhdGetBaseAngleXRad  
     dhdc.h, 39  
 dhdGetBaseAngleZDeg

dhdc.h, [39](#)  
dhdGetBaseAngleZRad  
dhdc.h, [39](#)  
dhdGetButton  
dhdc.h, [41](#)  
dhdGetButtonMask  
dhdc.h, [41](#)  
dhdGetComFreq  
dhdc.h, [42](#)  
dhdGetComMode  
dhdc.h, [42](#)  
dhdGetComponentVersionStr  
dhdc.h, [42](#)  
dhdGetDeltaEncoders  
dhdc.h, [43](#)  
dhdGetDeltaJacobian  
dhdc.h, [43](#)  
dhdGetDeltaJointAngles  
dhdc.h, [43](#)  
dhdGetDeviceAngleDeg  
dhdc.h, [44](#)  
dhdGetDeviceAngleRad  
dhdc.h, [44](#)  
dhdGetDeviceCount  
dhdc.h, [45](#)  
dhdGetDeviceID  
dhdc.h, [45](#)  
dhdGetEffectorMass  
dhdc.h, [45](#)  
dhdGetEnc  
dhdc.h, [46](#)  
dhdGetEncoder  
dhdc.h, [46](#)  
dhdGetEncRange  
dhdc.h, [47](#)  
dhdGetEncVelocities  
dhdc.h, [47](#)  
dhdGetForce  
dhdc.h, [48](#)  
dhdGetForceAndTorque  
dhdc.h, [48](#)  
dhdGetForceAndTorqueAndGripperForce  
dhdc.h, [49](#)  
dhdGetGripperAngleDeg  
dhdc.h, [49](#)  
dhdGetGripperAngleRad  
dhdc.h, [50](#)  
dhdGetGripperAngularVelocityDeg  
dhdc.h, [50](#)  
dhdGetGripperAngularVelocityRad  
dhdc.h, [51](#)  
dhdGetGripperEncoder  
dhdc.h, [52](#)  
dhdGetGripperFingerPos  
dhdc.h, [52](#)  
dhdGetGripperGap  
dhdc.h, [53](#)  
dhdGetGripperLinearVelocity  
dhdc.h, [53](#)  
dhdGetGripperThumbPos  
dhdc.h, [54](#)  
dhdGetJointAngleRange  
dhdc.h, [55](#)  
dhdGetJointAngles  
dhdc.h, [55](#)  
dhdGetJointVelocities  
dhdc.h, [55](#)  
dhdGetLinearVelocity  
dhdc.h, [56](#)  
dhdGetMaxForce  
dhdc.h, [56](#)  
dhdGetMaxGripperForce  
dhdc.h, [57](#)  
dhdGetMaxTorque  
dhdc.h, [57](#)  
dhdGetOrientationDeg  
dhdc.h, [58](#)  
dhdGetOrientationFrame  
dhdc.h, [59](#)  
dhdGetOrientationRad  
dhdc.h, [59](#)  
dhdGetPosition  
dhdc.h, [60](#)  
dhdGetPositionAndOrientationDeg  
dhdc.h, [60](#)  
dhdGetPositionAndOrientationFrame  
dhdc.h, [61](#)  
dhdGetPositionAndOrientationRad  
dhdc.h, [61](#)  
dhdGetSDKVersion  
dhdc.h, [62](#)  
dhdGetSDKVersionStr  
dhdc.h, [63](#)  
dhdGetSerialNumber  
dhdc.h, [63](#)  
dhdGetStatus  
dhdc.h, [63](#)  
dhdGetSystemCounter  
dhdc.h, [63](#)  
dhdGetSystemName  
dhdc.h, [64](#)  
dhdGetSystemRev  
dhdc.h, [64](#)  
dhdGetSystemType  
dhdc.h, [64](#)  
dhdGetTime  
dhdc.h, [65](#)  
dhdGetVelocityThreshold  
dhdc.h, [65](#)  
dhdGetVersion  
dhdc.h, [66](#)  
dhdGetVersionStr  
dhdc.h, [66](#)  
dhdGetWatchdog  
dhdc.h, [66](#)  
dhdGetWristEncoders  
dhdc.h, [67](#)  
dhdGetWristJacobian  
dhdc.h, [68](#)  
dhdGetWristJointAngles  
dhdc.h, [68](#)  
dhdGripperAngleRadToEncoder

dhdc.h, [68](#)  
 dhdGripperEncoderToAngleRad  
     dhdc.h, [69](#)  
 dhdGripperEncoderToGap  
     dhdc.h, [70](#)  
 dhdGripperForceToMotor  
     dhdc.h, [70](#)  
 dhdGripperGapToEncoder  
     dhdc.h, [71](#)  
 dhdGripperMotorToForce  
     dhdc.h, [72](#)  
 dhdHasActiveGripper  
     dhdc.h, [72](#)  
 dhdHasActiveWrist  
     dhdc.h, [73](#)  
 dhdHasBase  
     dhdc.h, [73](#)  
 dhdHasGripper  
     dhdc.h, [74](#)  
 dhdHasWrist  
     dhdc.h, [74](#)  
 dhdIsLeftHanded  
     dhdc.h, [75](#)  
 dhdJointAnglesToInertiaMatrix  
     dhdc.h, [76](#)  
 dhdKbGet  
     dhdc.h, [76](#)  
 dhdKbHit  
     dhdc.h, [76](#)  
 dhdOpen  
     dhdc.h, [76](#)  
 dhdOpenID  
     dhdc.h, [77](#)  
 dhdOpenSerial  
     dhdc.h, [77](#)  
 dhdOpenType  
     dhdc.h, [78](#)  
 dhdPreloadMot  
     dhdc.h, [78](#)  
 dhdPreset  
     dhdc.h, [79](#)  
 dhdReadConfigFromFile  
     dhdc.h, [79](#)  
 dhdReset  
     dhdc.h, [81](#)  
 dhdResetWrist  
     dhdc.h, [81](#)  
 dhdSetBaseAngleXDeg  
     dhdc.h, [81](#)  
 dhdSetBaseAngleXRad  
     dhdc.h, [82](#)  
 dhdSetBaseAngleZDeg  
     dhdc.h, [82](#)  
 dhdSetBaseAngleZRad  
     dhdc.h, [83](#)  
 dhdSetBrakes  
     dhdc.h, [83](#)  
 dhdSetBrk  
     dhdc.h, [84](#)  
 dhdSetComMode  
     dhdc.h, [84](#)  
 dhdSetComModePriority  
     dhdc.h, [85](#)  
 dhdSetDeltaJointTorques  
     dhdc.h, [85](#)  
 dhdSetDeltaMotor  
     dhdc.h, [86](#)  
 dhdSetDevice  
     dhdc.h, [86](#)  
 dhdSetDeviceAngleDeg  
     dhdc.h, [87](#)  
 dhdSetDeviceAngleRad  
     dhdc.h, [87](#)  
 dhdSetEffectorMass  
     dhdc.h, [88](#)  
 dhdSetForce  
     dhdc.h, [88](#)  
 dhdSetForceAndGripperForce  
     dhdc.h, [88](#)  
 dhdSetForceAndTorque  
     dhdc.h, [89](#)  
 dhdSetForceAndTorqueAndGripperForce  
     dhdc.h, [89](#)  
 dhdSetForceAndWristJointTorques  
     dhdc.h, [90](#)  
 dhdSetForceAndWristJointTorquesAndGripperForce  
     dhdc.h, [91](#)  
 dhdSetGravityCompensation  
     dhdc.h, [91](#)  
 dhdSetGripperMotor  
     dhdc.h, [92](#)  
 dhdSetMaxForce  
     dhdc.h, [92](#)  
 dhdSetMaxGripperForce  
     dhdc.h, [93](#)  
 dhdSetMaxTorque  
     dhdc.h, [94](#)  
 dhdSetMot  
     dhdc.h, [94](#)  
 dhdSetMotor  
     dhdc.h, [95](#)  
 dhdSetOutput  
     dhdc.h, [95](#)  
 dhdSetStandardGravity  
     dhdc.h, [96](#)  
 dhdSetTimeGuard  
     dhdc.h, [96](#)  
 dhdSetVelocityThreshold  
     dhdc.h, [96](#)  
 dhdSetVibration  
     dhdc.h, [97](#)  
 dhdSetWatchdog  
     dhdc.h, [97](#)  
 dhdSetWristJointTorques  
     dhdc.h, [98](#)  
 dhdSetWristMotor  
     dhdc.h, [98](#)  
 dhdSleep  
     dhdc.h, [99](#)  
 dhdStartThread  
     dhdc.h, [99](#)  
 dhdStop



- dhdc.h, [100](#)
- dhdUpdateEncoders
  - dhdc.h, [100](#)
- dhdWaitForReset
  - dhdc.h, [100](#)
- dhdWristEncodersToJointAngles
  - dhdc.h, [101](#)
- dhdWristEncoderToOrientation
  - dhdc.h, [101](#)
- dhdWristGravityJointTorques
  - dhdc.h, [102](#)
- dhdWristJointAnglesToEncoders
  - dhdc.h, [103](#)
- dhdWristJointAnglesToJacobian
  - dhdc.h, [103](#)
- dhdWristJointTorquesExtrema
  - dhdc.h, [104](#)
- dhdWristMotorToTorque
  - dhdc.h, [105](#)
- dhdWristOrientationToEncoder
  - dhdc.h, [105](#)
- dhdWristTorqueToMotor
  - dhdc.h, [106](#)