force dimension sdk

USER MANUAL

Haptic SDK
version 3.7.0

Force Dimension
Allée de la Petite Prairie 2
CH - 1260 Nyon
Switzerland

www.forcedimension.com

Contents

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 X

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
- WINCE7 must be defined for Microsoft Windows Compact Embedded 7 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 Delta Haptic Device 6DOF
  - DHD_DEVICE_DELTA6
- 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
  - DHD_DEVICE_SIGMA33P
  - DHD_DEVICE_SIGMA33P_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 forced 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_FORCEOFFCAUSE

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.

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()
- dhdStop()
- dhdGetComMode()
- dhdEnableForce()
- dhdEnableGripperForce()
- dhdGetSystemType()
- dhdGetSystemName()
- dhdGetVersion()
- dhdGetSDKVersion()
- dhdGetStatus()
- dhdGetDeviceAngleRad()
- dhdGetDeviceAngleDeg()
- dhdGetEffectorMass()
- dhdGetSystemCounter()
- dhdGetButton()
- dhdGetButtonMask()
- dhdSetOutput()
- dhdIsLeftHanded()
- dhdHasBase()
- dhdHasWrist()
- dhdHasActiveWrist()
- dhdHasGripper()
- dhdHasActiveGripper()
- dhdReset()
- dhdResetWrist()
- dhdWaitForReset()
- dhdSetStandardGravity()
- dhdSetGravityCompensation()
- dhdSetBrakes()
- dhdSetDeviceAngleRad()

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()
- dhdSetBrk()
- dhdGetDeltaJointAngles()
- dhdGetDeltaJacobian()
- dhdDeltaJointAnglesToJacobian()
- dhdDeltaJointTorquesExtrema()
- dhdDeltaGravityJointTorques()
- dhdSetDeltaJointTorques()
- dhdDeltaEncodersToJointAngles()
- dhdDeltaJointAnglesToEncoders()
- dhdGetWristJointAngles()
- dhdGetWristJacobian()
- dhdWristJointAnglesToJacobian()
- dhdWristJointTorquesExtrema()
- dhdWristGravityJointTorques()
- dhdSetWristJointTorques()
- dhdSetForceAndWristJointTorques()
- dhdSetForceAndWristJointTorquesAndGripperForce()
- dhdWristEncodersToJointAngles()
- dhdWristJointAnglesToEncoders()
- dhdGetJointAngles()
- 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.

2.1 References

Please use the following image when referencing the Force Dimension SDK in your own documentation:

Please use the following website when referencing Force Dimension products and services in your own documentation:

3 File Index

3.1 File List

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

4 File Documentation

4.1 dhdc.h File Reference

DHD header file.

Macros

- #define DHD_DEVICE_NONE 0
- #define DHD_DEVICE_DELTA3 63
- #define DHD_DEVICE_DELTA6 64
- #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_SIGMA33P 106
- #define DHD_DEVICE_SIGMA33P_LEFT 107
- #define DHD_ON 1
- #define DHD_OFF 0
- #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 16
- #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_FORCEOFFCAUSE 14
- #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_NULL_ARGUMENT,
  DHD_ERROR_REDUNDANT_FAIL, DHD_ERROR_NOT_ENABLED, DHD_ERROR_DEVICE_IN_USE }

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 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)
- const char *__SDK dhdGetSystemName (char ID=-1)
- int __SDK dhdGetVersion (double *ver, char ID=-1)
- void __SDK dhdGetSDKVersion (int *major, int *minor, int *release, int *revision)
- 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_WINDOWI↩NG, 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_WINDOW↩ING, 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_WINDOW↩ING, 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 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 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 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)

4.1.1 Detailed Description

DHD header file.

4.1.2 Macro Definition Documentation

4.1.2.1 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.

4.1.2.2 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.

4.1.2.3 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.

4.1.2.4 DHD_COM_MODE_VIRTUAL

```
#define DHD_COM_MODE_VIRTUAL 3
```

This mode is reported when connected to a virtual device.

4.1.2.5 DHD_DELTA_ENC_0

```
#define DHD_DELTA_ENC_0 0
```

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

4.1.2.6 DHD_DELTA_ENC_1

```
#define DHD_DELTA_ENC_1 1
```

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

4.1.2.7 DHD_DELTA_ENC_2

```
#define DHD_DELTA_ENC_2 2
```

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

4.1.2.8 DHD_DELTA_MOTOR_0

```
#define DHD_DELTA_MOTOR_0 0
```

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

4.1.2.9 DHD_DELTA_MOTOR_1

```
#define DHD_DELTA_MOTOR_1 1
```

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

4.1.2.10 DHD_DELTA_MOTOR_2

```
#define DHD_DELTA_MOTOR_2 2
```

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

4.1.2.11 DHD_DEVICE_CONTROLLER

```
#define DHD_DEVICE_CONTROLLER 81
```

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

4.1.2.12 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).

4.1.2.13 DHD_DEVICE_CUSTOM

```
#define DHD_DEVICE_CUSTOM 91
```

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

4.1.2.14 DHD_DEVICE_DELTA3

```
#define DHD_DEVICE_DELTA3 63
```

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

4.1.2.15 DHD_DEVICE_DELTA6

```
#define DHD_DEVICE_DELTA6 64
```

Device identifier for the Force Dimension DELTA.6 haptic device (USB version).

4.1.2.16 DHD_DEVICE_FALCON

```
#define DHD_DEVICE_FALCON 60
```

Device identifier for the Novint FALCON haptic device.

4.1.2.17 DHD_DEVICE_NONE

```
#define DHD_DEVICE_NONE 0
```

Device identifier returned when no device is connected.

4.1.2.18 DHD_DEVICE_OMEGA3

```
#define DHD_DEVICE_OMEGA3 33
```

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

4.1.2.19 DHD_DEVICE_OMEGA33

```
#define DHD_DEVICE_OMEGA33 34
```

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

4.1.2.20 DHD_DEVICE_OMEGA331

```
#define DHD_DEVICE_OMEGA331 35
```

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

4.1.2.21 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.

4.1.2.22 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.

4.1.2.23 DHD_DEVICE_SIGMA331

```
#define DHD_DEVICE_SIGMA331 104
```

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

4.1.2.24 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.

4.1.2.25 DHD_DEVICE_SIGMA33P

```
#define DHD_DEVICE_SIGMA33P 106
```

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

4.1.2.26 DHD_DEVICE_SIGMA33P_LEFT

```
#define DHD_DEVICE_SIGMA33P_LEFT 107
```

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

4.1.2.27 DHD_MAX_BUTTONS

```
#define DHD_MAX_BUTTONS 16
```

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

4.1.2.28 DHD_MAX_DOF

```
#define DHD_MAX_DOF 8
```

Maximum number of encoder channels that are available.

4.1.2.29 DHD_MAX_STATUS

```
#define DHD_MAX_STATUS 16
```

The length of the status array. See device status for details.

4.1.2.30 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.

4.1.2.31 DHD_OFF

```
#define DHD_OFF 0
```

Applies to the device status values.

4.1.2.32 DHD_ON

```
#define DHD_ON 1
```

Applies to the device status values.

4.1.2.33 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.

4.1.2.34 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.

4.1.2.35 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.

4.1.2.36 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.

4.1.2.37 DHD_STATUS_FORCEOFFCAUSE

```
#define DHD_STATUS_FORCEOFFCAUSE 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_FORCEOFF_NONE`: nothing has caused forces to be turned off yet
- `DHD_FORCEOFF_BUTTON`: the force button was pushed
- `DHD_FORCEOFF_VELOCITY`: the velocity threshold was reached
- `DHD_FORCEOFF_WATCHDOG`: the communication watchdog kicked in
- `DHD_FORCEOFF_SOFTWARE`: the software requested forces to be turned off, e.g. dhdEnableForce()
- `DHD_FORCEOFF_USBDISCN`: the USB cable was disconnected

- DHD_FORCEOFF_DEADMAN: the dead man switch was disconnected

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

4.1.2.38 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.

4.1.2.39 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.

4.1.2.40 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.

4.1.2.41 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.

4.1.2.42 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.

4.1.2.43 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.

4.1.2.44 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.

4.1.2.45 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.

4.1.2.46 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.

4.1.2.47 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.

4.1.2.48 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.

4.1.2.49 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.

4.1.2.50 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.

4.1.2.51 DHD_TIMEGUARD

```
#define DHD_TIMEGUARD 1
```

Return value used when the TimeGuard feature prevented an unnecessary communication with the device.

4.1.2.52 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.

4.1.2.53 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.

4.1.2.54 DHD_WRIST_ENC_0

```
#define DHD_WRIST_ENC_0 3
```

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

4.1.2.55 DHD_WRIST_ENC_1

```
#define DHD_WRIST_ENC_1 4
```

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

4.1.2.56 DHD_WRIST_ENC_2

```
#define DHD_WRIST_ENC_2 5
```

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

4.1.2.57 DHD_WRIST_MOTOR_0

```
#define DHD_WRIST_MOTOR_0 3
```

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

4.1.2.58 DHD_WRIST_MOTOR_1

```
#define DHD_WRIST_MOTOR_1 4
```

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

4.1.2.59 DHD_WRIST_MOTOR_2

```
#define DHD_WRIST_MOTOR_2 5
```

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

4.1.3 Enumeration Type Documentation

4.1.3.1 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 DHD functions.
DHD_ERROR	Undocumented error (for custom use).
DHD_ERROR_COM	Communication error between the dhd_controller and the host computer.
DHD_ERROR_DHC_BUSY	The device controller is busy and cannot perform the required task.
DHD_ERROR_NO_DRIVER_FOUND	A required device driver is not installed, please refer to your user manual installation section.
DHD_ERROR_NO_DEVICE_FOUND	No compatible Force Dimension device was detected.
DHD_ERROR_NOT_AVAILABLE	The command or feature is not available for a particular device or software release.
DHD_ERROR_TIMEOUT	The operation timed out.
DHD_ERROR_GEOMETRY	An error occurred within the device geometric model (expert mode only).
DHD_ERROR_EXPERT_MODE_DISABLED	The command or feature is not available because dhdEnableExpertMode() has not been called.
DHD_ERROR_NOT_IMPLEMENTED	The command or feature is currently not implemented.
DHD_ERROR_OUT_OF_MEMORY	Memory could not be allocated, please close some applications.
DHD_ERROR_DEVICE_NOT_READY	The 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/write the calibration data into the device memory.
DHD_ERROR_NULL_ARGUMENT	The function producing this error was given a null or invalid argument.
DHD_ERROR_REDUNDANT_FAIL	The redundant encoder integrity test failed.
DHD_ERROR_NOT_ENABLED	A particular feature is not enabled for the current device.
DHD_ERROR_DEVICE_IN_USE	The targeted device is already in use.

4.1.4 Function Documentation

4.1.4.1 dhdCalibrateWrist()

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

Trigger the WRIST calibration routine in the device controller.

Parameters

ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS
This feature only applies to the following device:

- DHD_DEVICE_DELTA6

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.2 dhdClose()

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

Close the connection to a particular device.

Parameters

ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.3 dhdConfigAngularVelocity()

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

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

Parameters

ms	**[default=DHD_VELOCITY_WINDOW]** time interval used to compute velocity [ms]
mode	**[default=DHD_VELOCITY_WINDOWING]** device ID (see velocity estimator modes section for details)
ID	**[default=-1]** device ID (see multiple devices section for details)

See also

velocity estimator for details
dhdGetAngularVelocityRad()
dhdGetAngularVelocityDeg()

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.4 dhdConfigGripperVelocity()

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

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

Parameters

ms	**[default=DHD_VELOCITY_WINDOW]** time interval used to compute velocity [ms]
mode	**[default=DHD_VELOCITY_WINDOWING]** device ID (see velocity estimator modes section for details)
ID	**[default=-1]** device ID (see multiple devices section for details)

See also

velocity estimator for details
dhdGetGripperLinearVelocity()
dhdGetGripperAngularVelocityRad()
dhdGetGripperAngularVelocityDeg()

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.5 dhdConfigLinearVelocity()

```
int __SDK dhdConfigLinearVelocity (
              int ms,
              int mode,
              char ID )
```

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

Parameters

ms	**[default=DHD_VELOCITY_WINDOW]** time interval used to compute velocity [ms]
mode	**[default=DHD_VELOCITY_WINDOWING]** device ID (see velocity estimator modes section for details)
ID	**[default=-1]** device ID (see multiple devices section for details)

See also

dhdGetLinearVelocity() and velocity estimator for details

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.6 dhdControllerSetDevice()

```
int __SDK dhdControllerSetDevice (
              int device,
              char ID )
```

If the connected device is a controller, this function lets you 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

device	the device type to use
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.7 dhdDeltaEncodersToJointAngles()

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

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

Parameters

enc0	DELTA encoder reading on axis 0
enc1	DELTA encoder reading on axis 1
enc2	DELTA encoder reading on axis 2
j0	**[out]** joint angle for axis 0 in [rad]
j1	**[out]** joint angle for axis 1 in [rad]
j2	**[out]** joint angle for axis 2 in [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.8 dhdDeltaEncoderToPosition()

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

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

Parameters

enc0	DELTA encoder reading on axis 0
enc1	DELTA encoder reading on axis 1
enc2	DELTA encoder reading on axis 2
px	**[out]** DELTA end-effector position on the X axis [m]
py	**[out]** DELTA end-effector position on the Y axis [m]
pz	**[out]** DELTA end-effector position on the Z axis [m]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.9 dhdDeltaForceToMotor()

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

This routine 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

fx	force on the DELTA end-effector on the X axis [N]
fy	force on the DELTA end-effector on the Y axis [N]
fz	force on the DELTA end-effector on the Z axis [N]
enc0	DELTA encoder reading on axis 0
enc1	DELTA encoder reading on axis 1
enc2	DELTA encoder reading on axis 2
mot0	**[out]** motor command on DELTA axis 0
mot1	**[out]** motor command on DELTA axis 1
mot2	**[out]** motor command on DELTA axis 2
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.10 dhdDeltaGravityJointTorques()

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

Compute 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

j0	joint angle for axis 0 in [rad]
j1	joint angle for axis 1 in [rad]
j2	joint angle for axis 2 in [rad]
q0	**out** gravity compensation joint torque on axis 0 in [Nm]
q1	**out** gravity compensation joint torque on axis 1 in [Nm]
q2	**out** gravity compensation joint torque on axis 2 in [Nm]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.11 dhdDeltaJointAnglesToEncoders()

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

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

Parameters

j0	joint angle for axis 0 in [rad]
j1	joint angle for axis 1 in [rad]
j2	joint angle for axis 2 in [rad]
enc0	**[out]** DELTA encoder reading on axis 0
enc1	**[out]** DELTA encoder reading on axis 1
enc2	**[out]** DELTA encoder reading on axis 2
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

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

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.12 dhdDeltaJointAnglesToJacobian()

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

Retrieve 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

j0	joint angle for axis 0 in [rad]
j1	joint angle for axis 1 in [rad]
j2	joint angle for axis 2 in [rad]
jcb	**[out]** device jacobian
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.13 dhdDeltaJointTorquesExtrema()

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

Compute 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

j0	joint angle for axis 0 in [rad]
j1	joint angle for axis 1 in [rad]
j2	joint angle for axis 2 in [rad]
minq	**out**array of minimum applicable joint torque in [Nm]
maxq	**out**array of maximum applicable joint torque in [Nm]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.14 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 routine 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

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

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for detail.

4.1.4.15 dhdDeltaPositionToEncoder()

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

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

Parameters

px	DELTA end-effector position on the X axis [m]
py	DELTA end-effector position on the Y axis [m]
pz	DELTA end-effector position on the Z axis [m]
enc0	**[out]** DELTA encoder reading on axis 0
enc1	**[out]** DELTA encoder reading on axis 1
enc2	**[out]** DELTA encoder reading on axis 2
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.16 dhdDisableExpertMode()

```
int __SDK dhdDisableExpertMode ( )
```

Disable the expert mode.

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.17 dhdEmulateButton()

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

Enable the button behavior emulation in the omega.7 gripper.

Parameters

val	DHD_ON to emulate button behavior, DHD_OFF to disable it
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.18 dhdEnableExpertMode()

```
int __SDK dhdEnableExpertMode ( )
```

Enable 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.

4.1.4.19 dhdEnableForce()

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

Enable the force mode in the device controller.

Parameters

val	DHD_ON to enable force, DHD_OFF to disable it
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.20 dhdEnableGripperForce()

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

Enable 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 dhdEnable↩ GripperForce() to have any effect.

Parameters

val	DHD_ON to enable gripper force, DHD_OFF to disable it
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.21 dhdEnableSimulator()

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

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

Parameters

on	**true** to enable, **false** to disable

4.1.4.22 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 ()

4.1.4.23 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.

4.1.4.24 dhdErrorGetStr()

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

Returns a brief string describing a given error code. See error management for details.

Parameters

error	error code

Returns

A pointer to a character array.

See also

dhdErrorGetLastStr ()

4.1.4.25 dhdGetAngularVelocityDeg()

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

Retrieve 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

wx	**[out]** angular velocity around the X axis [deg/s]
wy	**[out]** angular velocity around the Y axis [deg/s]
wz	**[out]** angular velocity around the Z axis [deg/s]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.26 dhdGetAngularVelocityRad()

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

Retrieve 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

wx	**[out]** angular velocity around the X axis [rad/s]
wy	**[out]** angular velocity around the Y axis [rad/s]
wz	**[out]** angular velocity around the Z axis [rad/s]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.27 dhdGetAvailableCount()

```
int __SDK dhdGetAvailableCount ( )
```

Return 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.

4.1.4.28 dhdGetBaseAngleXDeg()

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

Get the device base plate angle around the X axis.

Parameters

angle	**[out]** a pointer to a valid `double` to receive the device angle in [deg]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdGetBaseAngleXRad()

4.1.4.29 dhdGetBaseAngleXRad()

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

Get the device base plate angle around the X axis.

Parameters

angle	**[out]** a pointer to a valid `double` to receive the device angle in [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdGetBaseAngleXDeg()

4.1.4.30 dhdGetBaseAngleZDeg()

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

Get the device base plate angle around the vertical Z axis.

Parameters

angle	**[out]** a pointer to a valid `double` to receive the device angle in [deg]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdGetBaseAngleZRad()

4.1.4.31 dhdGetBaseAngleZRad()

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

Get the device base plate angle around the vertical Z axis.

Parameters

angle	**[out]** a pointer to a valid `double` to receive the device angle in [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdGetBaseAngleZDeg()

4.1.4.32 dhdGetButton()

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

Return the status of the button located on the end-effector.

Parameters

index	button index, 0 for the gripper button (up to DHD_MAX_BUTTONS)
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.33 dhdGetButtonMask()

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

Return the 32-bit binary mask of the device buttons.

Parameters

ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

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

4.1.4.34 dhdGetComFreq()

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

Return 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

ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

the refresh rate in [kHz], 0.0 otherwise.

4.1.4.35 dhdGetComMode()

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

Retrieve the COM operation mode on compatible devices.

Parameters

ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

the current COM operation mode on success, -1 otherwise.
See error management for details.

4.1.4.36 dhdGetDeltaEncoders()

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

Read all encoders values of the DELTA structure.

Parameters

enc0	**[out]** DELTA axis 0 encoder reading
enc1	**[out]** DELTA axis 1 encoder reading
enc2	**[out]** DELTA axis 2 encoder reading
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.37 dhdGetDeltaJacobian()

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

Retrieve 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

jacobian	**[out]** device jacobian
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

4.1.4.38 dhdGetDeltaJointAngles()

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

Retrieve the joint angles in [rad] for the DELTA structure.

Parameters

j0	**[out]** joint angle for axis 0 in [rad]
j1	**[out]** joint angle for axis 1 in [rad]
j2	**[out]** joint angle for axis 2 in [rad]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.39 dhdGetDeviceAngleDeg()

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

Get the device base plate angle around the Y axis.

Parameters

angle	**[out]** a pointer to a valid `double` to receive the device angle in [deg]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdGetDeviceAngleRad()

4.1.4.40 dhdGetDeviceAngleRad()

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

Get the device base plate angle around the Y axis.

Parameters

angle	**[out]** a pointer to a valid `double` to receive the device angle in [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdGetDeviceAngleDeg()

4.1.4.41 dhdGetDeviceCount()

```
int __SDK dhdGetDeviceCount ( )
```

Return 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.

4.1.4.42 dhdGetDeviceID()

```
int __SDK dhdGetDeviceID ( )
```

Return the ID of the current default device.

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.43 dhdGetEffectorMass()

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

Get the mass of the end-effector currently defined for a device. The gripper mass is used in the gravity compensation feature.

Parameters

mass	a pointer to the actual end-effector mass in [kg]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.44 dhdGetEnc()

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

Get selective encoder values into encoder array. Particularly useful when using the the generic controller directly, without a device model attached.

Parameters

enc	**out** encoder values array
mask	**[default=0xff]** bitwise mask of which encoders should be read in
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.45 dhdGetEncoder()

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

Read a single encoder value from the haptic device.

Parameters

index	the encoder index number as defined by DHD_MAX_DOF
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.46 dhdGetEncVelocities()

```
int __SDK dhdGetEncVelocities (
              double v[DHD_MAX_DOF],
              char ID )
```

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

Parameters

v	**out**array of joint angle velocities in [rad/s]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.47 dhdGetForce()

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

Retrieve the force vector applied to the end-effector.

Parameters

fx	**[out]** force on the X axis in [N]
fy	**[out]** force on the Y axis in [N]
fz	**[out]** force on the Z axis in [N]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.48 dhdGetForceAndTorque()

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

Retrieve the force and torque vectors applied to the device end-effector.

Parameters

fx	**[out]** force on the X axis in [N]
fy	**[out]** force on the Y axis in [N]
fz	**[out]** force on the Z axis in [N]
tx	**[out]** torque around the X axis in [Nm]
ty	**[out]** torque around the Y axis in [Nm]
tz	**[out]** torque around the Z axis in [Nm]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.49 dhdGetForceAndTorqueAndGripperForce()

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

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

Parameters

fx	**[out]** force on the X axis in [N]
fy	**[out]** force on the Y axis in [N]
fz	**[out]** force on the Z axis in [N]
tx	**[out]** torque around the X axis in [Nm]
ty	**[out]** torque around the Y axis in [Nm]
tz	**[out]** torque around the Z axis in [Nm]
f	**[out]** force on the gripper in [N]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.50 dhdGetGripperAngleDeg()

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

Get the gripper opening angle in degrees.

Parameters

a	**[out]** gripper opening [deg]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.51 dhdGetGripperAngleRad()

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

Get the gripper opening angle in radians.

Parameters

a	**[out]** gripper opening [rad]
ID	**[default=-1]** device ID (see multiple devices 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 on success, -1 otherwise.
See error management for details.

4.1.4.52 dhdGetGripperAngularVelocityDeg()

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

Retrieve 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

wg	**[out]** gripper angular velocity [deg/s]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.53 dhdGetGripperAngularVelocityRad()

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

Retrieve 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

wg	**[out]** gripper angular velocity [rad/s]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.54 dhdGetGripperEncoder()

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

Read the encoder value of the force gripper.

Parameters

enc	**[out]** gripper encoder reading
ID	**[default=-1]** device ID (see multiple devices 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

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.55 dhdGetGripperFingerPos()

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

Read the position in Cartesian coordinates of forefinger rest location of the force gripper structure if present.

Parameters

px	**[out]** gripper finger X coord
py	**[out]** gripper finger Y coord
pz	**[out]** gripper finger Z coord
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.56 dhdGetGripperGap()

```
int __SDK dhdGetGripperGap (
              double * g,
              char ID )
```

Get the gripper opening distance in meters.

Parameters

g	**[out]** gripper opening [m]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.57 dhdGetGripperLinearVelocity()

```
int __SDK dhdGetGripperLinearVelocity (
              double * vg,
              char ID )
```

Retrieve the estimated instantaneous linear velocity of the gripper in [m/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

vg	**[out]** gripper linear velocity [m/s]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.58 dhdGetGripperThumbPos()

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

Read the position in Cartesian coordinates of thumb rest location of the force gripper structure if present.

Parameters

px	**[out]** gripper thumb X coord
py	**[out]** gripper thumb Y coord
pz	**[out]** gripper thumb Z coord
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.59 dhdGetJointAngles()

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

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

Parameters

j	**out**array of joint angles in [rad]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.60 dhdGetJointVelocities()

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

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

Parameters

v	**out**array of joint angle velocities in [rad/s]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.61 dhdGetLinearVelocity()

```
int __SDK dhdGetLinearVelocity (
            double * vx,
            double * vy,
```

```
            double * vz,
            char ID )
```

Retrieve 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

vx	**[out]** velocity along the X axis [m/s]
vy	**[out]** velocity along the Y axis [m/s]
vz	**[out]** velocity along the Z axis [m/s]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.62 dhdGetOrientationDeg()

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

For the DHD_DEVICE_DELTA6 devices, retrieve the Euler angles of the WRIST structure (around XYZ). For all other devices, retrieve 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

oa	**[out]** device orientation around the X axis in [deg]
ob	**[out]** device orientation around the Y axis in [deg]
og	**[out]** device orientation around the Z axis in [deg]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

This feature only applies to the following devices:

- DHD_DEVICE_DELTA6
- DHD_DEVICE_OMEGA33

- DHD_DEVICE_OMEGA33_LEFT
- DHD_DEVICE_OMEGA331
- DHD_DEVICE_OMEGA331_LEFT
- DHD_DEVICE_SIGMA331
- DHD_DEVICE_SIGMA331_LEFT
- DHD_DEVICE_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

See also

dhdGetOrientationRad()

4.1.4.63 dhdGetOrientationFrame()

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

Retrieve the rotation matrix of the WRIST structure. The identity matrix is returned for devices that do not support orientations.

Parameters

matrix	**[out]** orientation matrix frame
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

4.1.4.64 dhdGetOrientationRad()

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

For the DHD_DEVICE_DELTA6 devices, retrieve the Euler angles of the WRIST structure (around XYZ). For all other devices, retrieve 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

oa	**[out]** device orientation around the X axis in [rad]
ob	**[out]** device orientation around the Y axis in [rad]

Parameters

og	**[out]** device orientation around the Z axis in [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

This feature only applies to the following devices:

- DHD_DEVICE_DELTA6
- DHD_DEVICE_OMEGA33
- DHD_DEVICE_OMEGA33_LEFT
- DHD_DEVICE_OMEGA331
- DHD_DEVICE_OMEGA331_LEFT
- DHD_DEVICE_SIGMA331
- DHD_DEVICE_SIGMA331_LEFT
- DHD_DEVICE_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

See also

dhdGetOrientationDeg()

4.1.4.65 dhdGetPosition()

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

Retrieve 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

px	**[out]** device position on the X axis in [m]
py	**[out]** device position on the Y axis in [m]
pz	**[out]** device position on the Z axis in [m]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

Examples:

hello_world.cpp.

4.1.4.66 dhdGetPositionAndOrientationDeg()

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

Retrieve the position and orientation of the end-effector in Cartesian coordinates. For the DHD_DEVICE_DELTA6 devices, the orientation is expressed as the Euler angles of the WRIST structure (around XYZ). For all other devices, 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

px	**[out]** device position on the X axis in [m]
py	**[out]** device position on the Y axis in [m]
pz	**[out]** device position on the Z axis in [m]
oa	**[out]** device orientation around the X axis in [deg]
ob	**[out]** device orientation around the Y axis in [deg]
og	**[out]** device orientation around the Z axis in [deg]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

4.1.4.67 dhdGetPositionAndOrientationFrame()

```
int __SDK dhdGetPositionAndOrientationFrame (
              double * px,
              double * py,
              double * pz,
              double matrix[3][3],
              char ID )
```

Retrieve 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

px	**[out]** device position on the X axis in [m]
py	**[out]** device position on the Y axis in [m]
pz	**[out]** device position on the Z axis in [m]
matrix	**[out]** orientation matrix frame
ID	[default=-1] device ID (see multiple devices section for details)

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

4.1.4.68 dhdGetPositionAndOrientationRad()

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

Retrieve the position and orientation of the end-effector in Cartesian coordinates. For the DHD_DEVICE_DELTA6 devices, the orientation is expressed as the Euler angles of the WRIST structure (around XYZ). For all other devices, 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

px	**[out]** device position on the X axis in [m]
py	**[out]** device position on the Y axis in [m]
pz	**[out]** device position on the Z axis in [m]
oa	**[out]** device orientation around the X axis in [rad]
ob	**[out]** device orientation around the Y axis in [rad]
og	**[out]** device orientation around the Z axis in [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

4.1.4.69 dhdGetSDKVersion()

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

Return the SDK complete set of version numbers.

Parameters

major	major version number
minor	minor version number
release	release number
revision	revision number

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.70 dhdGetSerialNumber()

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

Return the device serial number.

Parameters

sn	**[out]** a pointer to a valid `unsigned char` to receive the device serial number
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.71 dhdGetStatus()

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

Returns the status vector of the haptic device. The status is described in the status section.

Parameters

status	**[out]** an array that will receive the status vector
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.72 dhdGetSystemCounter()

```
ulong __SDK dhdGetSystemCounter ( )
```

Returns a timestamp computed from the high-resolution system counter, expressed in microseconds. This function is deprecated, please usr dhdGetTime() instead.

Returns

A timestamp in [us].

See also

See dhdGetTime()

4.1.4.73 dhdGetSystemName()

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

Return 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 multiple devices section for details)

Returns

The device type string on success, NULL otherwise.
See error management for details.

4.1.4.74 dhdGetSystemType()

```
int __SDK dhdGetSystemType (
            char ID )
```

Return 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 multiple devices section for details)

Returns

The device type on success, -1 otherwise.
See error management for details.

4.1.4.75 dhdGetTime()

```
double __SDK dhdGetTime ( )
```

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.

Returns

The current time in [s].

4.1.4.76 dhdGetVelocityThreshold()

```
int __SDK dhdGetVelocityThreshold (
            uint * val,
            char ID )
```

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 **NOT** to modify factory settings
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.77 dhdGetVersion()

```
int __SDK dhdGetVersion (
              double * ver,
              char ID )
```

Return 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

ver	**[out]** pointer to a variable that will receive the controller release version number
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.78 dhdGetWatchdog()

```
int __SDK dhdGetWatchdog (
              uchar * val,
              char ID )
```

Get the watchdog duration in multiples of 125 microseconds on compatible devices.

Parameters

val	**[out]**watchdog duration in multiples of 125 [us]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.79 dhdGetWristEncoders()

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

Read the encoders values of the WRIST structure.

Parameters

enc0	**[out]** WRIST axis 0 encoder reading
enc1	**[out]** WRIST axis 1 encoder reading
enc2	**[out]** WRIST axis 2 encoder reading
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS
This feature only applies to the following devices:

- DHD_DEVICE_DELTA6
- DHD_DEVICE_OMEGA33
- DHD_DEVICE_OMEGA33_LEFT
- DHD_DEVICE_OMEGA331
- DHD_DEVICE_OMEGA331_LEFT
- DHD_DEVICE_SIGMA331
- DHD_DEVICE_SIGMA331_LEFT
- DHD_DEVICE_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

4.1.4.80 dhdGetWristJacobian()

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

Retrieve 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

jacobian	**[out]** device jacobian
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

4.1.4.81 dhdGetWristJointAngles()

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

Retrieve the joint angles in [rad] for the WRIST structure.

Parameters

j0	**[out]** joint angle for axis 0 in [rad]
j1	**[out]** joint angle for axis 1 in [rad]
j2	**[out]** joint angle for axis 2 in [rad]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.82 dhdGripperAngleRadToEncoder()

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

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

Parameters

a	gripper opening in [rad]
enc	**[out]** gripper encoder reading
ID	**[default=-1]** device ID (see multiple devices 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

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.83 dhdGripperEncoderToAngleRad()

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

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

Parameters

enc	gripper encoder reading
a	**[out]** gripper opening [rad]
ID	**[default=-1]** device ID (see multiple devices 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

Returns

0 or DHD_TIMEGUARD on success, -1 otherwise.
See error management for details.

4.1.4.84 dhdGripperEncoderToGap()

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

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

Parameters

enc	gripper encoder reading
gap	**[out]** gripper opening [m]
ID	**[default=-1]** device ID (see multiple devices 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

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.85 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 routine computes and returns the corresponding motor command.

Parameters

frc	force on the gripper end-effector [N]
mot	**[out]** motor command on gripper axis
enc	encoder reading for wrist (at indices 0,1,2) and gripper (at index 3)
ID	**[default=-1]** device ID (see multiple devices 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

Returns

0 or DHD_MOTOR_SATURATED on success, -1 otherwise.
See error management for details.

4.1.4.86 dhdGripperGapToEncoder()

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

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

Parameters

gap	gripper opening in [m]
enc	**[out]** gripper encoder reading
ID	**[default=-1]** device ID (see multiple devices 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

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.87 dhdGripperMotorToForce()

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

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

Parameters

mot	motor command on gripper axis
frc	**[out]** force on the gripper end-effector [N]
enc	encoder reading for wrist (at indices 0,1,2) and gripper (at index 3)
ID	**[default=-1]** device ID (see multiple devices 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

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.88 dhdHasActiveGripper()

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

Returns **true** if the device has an active gripper, **false** otherwise.

Parameters

ID	**[default=-1]** device ID (see multiple devices 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_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

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

4.1.4.89 dhdHasActiveWrist()

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

Returns **true** if the device has an active wrist, **false** otherwise.

Parameters

ID	**[default=-1]** device ID (see multiple devices 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_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

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

4.1.4.90 dhdHasBase()

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

Returns **true** if the device has a base, **false** otherwise.

Parameters

ID	**[default=-1]** device ID (see multiple devices section for details)

Note

This feature only applies to the following devices:

- DHD_DEVICE_DELTA3
- DHD_DEVICE_DELTA6
- 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_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT
- DHD_DEVICE_FALCON

Returns

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

4.1.4.91 dhdHasGripper()

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

Returns **true** if the device has a gripper, **false** otherwise.

Parameters

ID	**[default=-1]** device ID (see multiple devices 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_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

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

4.1.4.92 dhdHasWrist()

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

Returns **true** if the device has a wrist, **false** otherwise.

Parameters

ID	**[default=-1]** device ID (see multiple devices section for details)

Note

This feature only applies to the following devices:

- DHD_DEVICE_DELTA6
- DHD_DEVICE_OMEGA33
- DHD_DEVICE_OMEGA33_LEFT
- DHD_DEVICE_OMEGA331
- DHD_DEVICE_OMEGA331_LEFT
- DHD_DEVICE_SIGMA331
- DHD_DEVICE_SIGMA331_LEFT
- DHD_DEVICE_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

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

4.1.4.93 dhdIsLeftHanded()

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

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

Parameters

ID	**[default=-1]** device ID (see multiple devices 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_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

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

4.1.4.94 dhdJointAnglesToInertiaMatrix()

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

Retrieve 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

j	array of joint angles in [rad]
inertia	**[out]** device inertia matrix
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.95 dhdKbGet()

```
char __SDK dhdKbGet ( )
```

Retrieve a character from the keyboard (OS independent).

Returns

The matching keyboard character.

4.1.4.96 dhdKbHit()

```
bool __SDK dhdKbHit ( )
```

Check keyboard for a key hit (OS independent).

Returns

true on key pressed, false otherwise.

4.1.4.97 dhdOpen()

```
int __SDK dhdOpen ( )
```

Open 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

dhdOpenID()

Examples:

hello_world.cpp, and single_device.cpp.

4.1.4.98 dhdOpenID()

```
int __SDK dhdOpenID (
            char ID )
```

Open a connection to one particular device connected to the system. The order in which devices are opened persists until devices are added or removed.

Parameters

ID	the device ID (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

dhdOpen()

Examples:

multiple_devices.cpp.

4.1.4.99 dhdOpenSerial()

```
int __SDK dhdOpenSerial (
            int serial )
```

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

Parameters

serial	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()

4.1.4.100 dhdOpenType()

```
int __SDK dhdOpenType (
            int type )
```

Open 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

type	requested system Device Types 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()

4.1.4.101 dhdPreloadMot()

```
int __SDK dhdPreloadMot (
            ushort mot[DHD_MAX_DOF],
            uchar mask,
            char ID )
```

Program motor commands to a selection of motor channels. Unlike dhdSetMot(), this function saves the requested commands internally for later application by calling dhdSetForce() and the likes.

Parameters

mot	motor values array
mask	**[default=0xff]** bitwise mask of which motor should be set
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.102 dhdPreset()

```
int __SDK dhdPreset (
            int val[DHD_MAX_DOF],
            uchar mask,
            char ID )
```

Set selected encoder offsets to a given value. Intended for use with the the generic controller when no RESET button is available.

Parameters

val	motor values array
mask	bitwise mask of which encoder should be set
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.103 dhdReadConfigFromFile()

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

Load 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

filename	configuration file
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.104 dhdReset()

```
int __SDK dhdReset (
            char ID )
```

Puts the device in RESET mode.

Parameters

ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.105 dhdResetWrist()

```
int __SDK dhdResetWrist (
            char ID )
```

Puts the DELTA.6 active device WRIST extension in RESET mode.

Parameters

ID	**[default=-1]** device ID (see multiple devices section for details)

Note

This feature only applies to the following device:

- DHD_DEVICE_DELTA6

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.106 dhdSetBaseAngleXDeg()

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

Set 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

angle	device base plate angle around X [deg]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdSetBaseAngleXRad()

4.1.4.107 dhdSetBaseAngleXRad()

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

Set 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

angle	device base plate angle around X [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdSetBaseAngleXDeg()

4.1.4.108 dhdSetBaseAngleZDeg()

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

Set 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

angle	device base plate angle around Z [deg]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdSetBaseAngleZRad()

4.1.4.109 dhdSetBaseAngleZRad()

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

Set 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

angle	device base plate angle around Z [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdSetBaseAngleZDeg()

4.1.4.110 dhdSetBrakes()

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

Enable/disable the device electromagnetic brakes.

Parameters

val	desired state of the brakes (DHD_ON or DHD_OFF)
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.111 dhdSetBrk()

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

Set brakes electromagnetic brakes status on selective motor groups. Only applies when using the generic controller directly, without a device model attached.

Parameters

mask	**[default=0xff]** bitwise mask of which motor group should the brakes be set on.
ID	**[default=-1]** device ID (see multiple devices 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. I 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.

4.1.4.112 dhdSetComMode()

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

Set the COM operation mode on compatible devices.

Parameters

mode	desired COM operation mode
ID	**[default=-1]** device ID (see multiple devices 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()

4.1.4.113 dhdSetComModePriority()

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

Set the priority of the thread (if any) that supports the current COM operation mode on compatible devices. Currently unused.

Parameters

priority	desired thread priority
ID	**[default=-1]** device ID (see multiple devices 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()

4.1.4.114 dhdSetDeltaJointTorques()

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

Set all joint torques of the DELTA structure.

Parameters

t0	DELTA axis 0 torque command [Nm]
t1	DELTA axis 1 torque command [Nm]
t2	DELTA axis 2 torque command [Nm]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.115 dhdSetDeltaMotor()

```
int __SDK dhdSetDeltaMotor (
```

```
            ushort mot0,
            ushort mot1,
            ushort mot2,
            char ID )
```

Set desired motor commands to the amplifier channels commanding the DELTA motors.

Parameters

mot0	DELTA axis 0 motor command
mot1	DELTA axis 1 motor command
mot2	DELTA axis 2 motor command
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.116 dhdSetDevice()

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

Select 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

ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.117 dhdSetDeviceAngleDeg()

```
int __SDK dhdSetDeviceAngleDeg (
            double angle,
            char ID )
```

Set 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

angle	device base plate angle [deg]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdSetDeviceAngleRad()

4.1.4.118 dhdSetDeviceAngleRad()

```
int __SDK dhdSetDeviceAngleRad (
            double angle,
            char ID )
```

Set 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

angle	device base plate angle [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

See also

dhdSetDeviceAngleDeg()

4.1.4.119 dhdSetEffectorMass()

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

Define 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

mass	the actual end-effector mass in [kg]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.120 dhdSetForce()

```
int __SDK dhdSetForce (
```

```
            double fx,
            double fy,
            double fz,
            char ID )
```

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

Parameters

fx	force on the X axis in [N]
fy	force on the Y axis in [N]
fz	force on the Z axis in [N]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.121 dhdSetForceAndGripperForce()

```
int __SDK dhdSetForceAndGripperForce (
            double fx,
            double fy,
            double fz,
            double fg,
            char ID )
```

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

Parameters

fx	translation force along X axis
fy	translation force along Y axis
fz	translation force along Z axis
fg	grasping force
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 or DHD_MOTOR_SATURATED on success, -1 otherwise.
See error management for details.

4.1.4.122 dhdSetForceAndTorque()

```
int __SDK dhdSetForceAndTorque (
            double fx,
            double fy,
            double fz,
            double tx,
```

```
            double ty,
            double tz,
            char ID )
```

Set the desired force and torque vectors to be applied to the device end-effector.

Parameters

fx	force on the X axis in [N]
fy	force on the Y axis in [N]
fz	force on the Z axis in [N]
tx	torque around the X axis in [Nm]
ty	torque around the Y axis in [Nm]
tz	torque around the Z axis in [Nm]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 or DHD_MOTOR_SATURATED on success, -1 otherwise.
See error management for details.

4.1.4.123 dhdSetForceAndTorqueAndGripperForce()

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

Set 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

fx	force on the X axis in [N]
fy	force on the Y axis in [N]
fz	force on the Z axis in [N]
tx	torque around the X axis in [Nm]
ty	torque around the Y axis in [Nm]
tz	torque around the Z axis in [Nm]
fg	gripper force in [N]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 or DHD_MOTOR_SATURATED on success, -1 otherwise.
See error management for details.

4.1.4.124 dhdSetForceAndWristJointTorques()

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

Set Cartesian force and wrist joint torques.

Parameters

fx	translation force along X axis [N]
fy	translation force along Y axis [N]
fz	translation force along Z axis [N]
t0	WRIST axis 0 torque command [Nm]
t1	WRIST axis 1 torque command [Nm]
t2	WRIST axis 2 torque command [Nm]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.125 dhdSetForceAndWristJointTorquesAndGripperForce()

```
int __SDK dhdSetForceAndWristJointTorquesAndGripperForce (
              double fx,
              double fy,
              double fz,
              double t0,
              double t1,
              double t2,
              double fg,
              char ID )
```

Set Cartesian force, wrist joint torques and gripper force.

Parameters

fx	translation force along X axis [N]
fy	translation force along Y axis [N]
fz	translation force along Z axis [N]

Parameters

t0	WRIST axis 0 torque command [Nm]
t1	WRIST axis 1 torque command [Nm]
t2	WRIST axis 2 torque command [Nm]
fg	gripper force in [N]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.126 dhdSetGravityCompensation()

```
int __SDK dhdSetGravityCompensation (
            int val,
            char ID )
```

Enable/disable gravity compensation.

Parameters

val	desired state of the gravity compensation feature (DHD_ON or DHD_OFF)
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.127 dhdSetGripperMotor()

```
int __SDK dhdSetGripperMotor (
            ushort mot,
            char ID )
```

Set a desired motor command to the force gripper.

Parameters

mot	gripper motor command
ID	**[default=-1]** device ID (see multiple devices 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

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.128 dhdSetMot()

```
int __SDK dhdSetMot (
            ushort mot[DHD_MAX_DOF],
            uchar mask,
            char ID )
```

Program motor commands to a selection of motor channels. Particularly useful when using the generic controller directly, without a device model attached.

Parameters

mot	motor values array
mask	**[default=0xff]** bitwise mask of which motor should be set
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.129 dhdSetMotor()

```
int __SDK dhdSetMotor (
            int index,
            ushort val,
            char ID )
```

Program a command to a single motor channel.

Parameters

index	the motor index number as defined by DHD_MAX_DOF
val	the motor DAC value
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.130 dhdSetOutput()

```
int __SDK dhdSetOutput (
            uint output,
            char ID )
```

Set the user programmable output bits on devices that support it.

Parameters

output	a bitwise mask that toggles the programmable output bits
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

This feature only applies to the following devices:

- DHD_DEVICE_DELTA3
- DHD_DEVICE_DELTA6
- DHD_DEVICE_SIGMA331
- DHD_DEVICE_SIGMA331_LEFT
- DHD_DEVICE_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.131 dhdSetStandardGravity()

```
int __SDK dhdSetStandardGravity (
            double g,
            char ID )
```

Set the standard gravity constant used in gravity compensation. By default, the constant is set to 9.81 m/s^2.

Parameters

g	standard gravity constant [m/s^2]
ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.132 dhdSetTimeGuard()

```
int __SDK dhdSetTimeGuard (
            int us,
            char ID )
```

Enable/disable the TimeGuard feature with an arbitrary minimum period.

Parameters

us	minimum refresh period in [us] a value of 0 disables the TimeGuard feature, while a value of -1 resets the default value (recommended)
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.133 dhdSetVelocityThreshold()

```
int __SDK dhdSetVelocityThreshold (
            uint val,
            char ID )
```

Adjust 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

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

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.134 dhdSetVibration()

```
int __SDK dhdSetVibration (
            double freq,
            double amplitude,
            int type,
            char ID )
```

Apply a vibration to the end-effector. The vibration is a added to the force requested by dhdSetForce() and the like. The vibration application mechanism depends on the specific device type, and is currently only available on devices with dedicated vibration actuators.

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.135 dhdSetWatchdog()

```
int __SDK dhdSetWatchdog (
            uchar val,
            char ID )
```

Set 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

val	watchdog duration in multiples of 125 [us]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

See also

dhdGetWatchdog()

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.136 dhdSetWristJointTorques()

```
int __SDK dhdSetWristJointTorques (
            double t0,
            double t1,
            double t2,
            char ID )
```

Set all joint torques of the WRIST structure.

Parameters

t0	WRIST axis 0 torque command [Nm]

Parameters

t1	WRIST axis 1 torque command [Nm]
t2	WRIST axis 2 torque command [Nm]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.137 dhdSetWristMotor()

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

Set desired motor commands to the amplifier channels commanding the WRIST motors.

Parameters

mot0	WRIST axis 0 motor command
mot1	WRIST axis 1 motor command
mot2	WRIST axis 2 motor command
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS
This feature only applies to the following devices:

- DHD_DEVICE_DELTA6
- DHD_DEVICE_SIGMA331
- DHD_DEVICE_SIGMA331_LEFT
- DHD_DEVICE_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.138 dhdSleep()

```
void __SDK dhdSleep (
              double sec )
```

Sleep for a given period of time (OS independent).

Parameters

sec	sleep period in [s]

4.1.4.139 dhdStartThread()

```
int __SDK dhdStartThread (
            void * funcvoid *,
            void * arg,
            int priority )
```

Create 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

func	function to run in the thread
arg	optional pointer to an argument passed to the thread function
priority	priority given to the thread (see the multi-threading 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.

4.1.4.140 dhdStop()

```
int __SDK dhdStop (
            char ID )
```

Stop the device. This routine disables the force on the haptic device and puts it into BRAKE mode.

Parameters

ID	**[default=-1]** device ID (see multiple devices section for details)

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.141 dhdUpdateEncoders()

```
int __SDK dhdUpdateEncoders (
            char ID )
```

Force 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

ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.142 dhdWaitForReset()

```
int __SDK dhdWaitForReset (
              int timeout,
              char ID )
```

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

timeout	[optional] maximum time to wait for calibration in [ms]
ID	**[default=-1]** device ID (see multiple devices 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.

4.1.4.143 dhdWristEncodersToJointAngles()

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

This routine computes and returns the wrist joint angles for a given set of encoder readings.

Parameters

enc0	WRIST encoder reading on axis 0
enc1	WRIST encoder reading on axis 1
enc2	WRIST encoder reading on axis 2
j0	**[out]** joint angle for axis 0 in [rad]

Parameters

j1	**[out]** joint angle for axis 1 in [rad]
j2	**[out]** joint angle for axis 2 in [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.144 dhdWristEncoderToOrientation()

```
int __SDK dhdWristEncoderToOrientation (
            int enc0,
            int enc1,
            int enc2,
            double * oa,
            double * ob,
            double * og,
            char ID )
```

For the DHD_DEVICE_DELTA6 devices, compute the Euler angles of the WRIST structure (around XYZ). For all other devices, compute 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

enc0	WRIST encoder reading on axis 0
enc1	WRIST encoder reading on axis 1
enc2	WRIST encoder reading on axis 2
oa	**[out]** WRIST end-effector orientation around the X axis [rad]
ob	**[out]** WRIST end-effector orientation around the Y axis [rad]
og	**[out]** WRIST end-effector orientation around the Z axis [rad]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS
This feature only applies to the following devices:

- DHD_DEVICE_DELTA6
- DHD_DEVICE_OMEGA33
- DHD_DEVICE_OMEGA33_LEFT
- DHD_DEVICE_OMEGA331
- DHD_DEVICE_OMEGA331_LEFT
- DHD_DEVICE_SIGMA331
- DHD_DEVICE_SIGMA331_LEFT

- DHD_DEVICE_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.145 dhdWristGravityJointTorques()

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

Compute 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

j0	joint angle for axis 0 in [rad]
j1	joint angle for axis 1 in [rad]
j2	joint angle for axis 2 in [rad]
q0	**out** gravity compensation joint torque on axis 0 in [Nm]
q1	**out** gravity compensation joint torque on axis 1 in [Nm]
q2	**out** gravity compensation joint torque on axis 2 in [Nm]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.146 dhdWristJointAnglesToEncoders()

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

This routine computes and returns the wrist encoder readings for a given set of joint angles.

Parameters

j0	joint angle for axis 0 in [rad]
j1	joint angle for axis 1 in [rad]
j2	joint angle for axis 2 in [rad]
enc0	**[out]** WRIST encoder reading on axis 0
enc1	**[out]** WRIST encoder reading on axis 1
enc2	**[out]** WRIST encoder reading on axis 2
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.147 dhdWristJointAnglesToJacobian()

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

Retrieve 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

j0	joint angle for axis 0 in [rad]
j1	joint angle for axis 1 in [rad]
j2	joint angle for axis 2 in [rad]
jcb	**[out]** device jacobian
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.148 dhdWristJointTorquesExtrema()

```
int __SDK dhdWristJointTorquesExtrema (
              double j0,
              double j1,
```

```
            double j2,
            double minq[3],
            double maxq[3],
            char ID )
```

Compute 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

j0	joint angle for axis 0 in [rad]
j1	joint angle for axis 1 in [rad]
j2	joint angle for axis 2 in [rad]
minq	**out**array of minimum applicable joint torque in [Nm]
maxq	**out**array of maximum applicable joint torque in [Nm]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.149 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 routine 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

mot0	motor command around WRIST axis 0
mot1	motor command around WRIST axis 1
mot2	motor command around WRIST axis 2
enc0	WRIST encoder reading around axis 0
enc1	WRIST encoder reading around axis 1
enc2	WRIST encoder reading around axis 2
tx	**[out]** torque on the WRIST end-effector around the X axis [Nm]
ty	**[out]** torque on the WRIST end-effector around the Y axis [Nm]
tz	**[out]** torque on the WRIST end-effector around the Z axis [Nm]
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS
This feature only applies to the following devices:

- DHD_DEVICE_DELTA6
- DHD_DEVICE_SIGMA331
- DHD_DEVICE_SIGMA331_LEFT
- DHD_DEVICE_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.150 dhdWristOrientationToEncoder()

```
int __SDK dhdWristOrientationToEncoder (
            double oa,
            double ob,
            double og,
            int * enc0,
            int * enc1,
            int * enc2,
            char ID )
```

For the DHD_DEVICE_DELTA6 devices, compute the encoder values from the Euler angles of the WRIST structure (around XYZ). For all other devices, compute 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

oa	WRIST end-effector orientation around the X axis [rad]
ob	WRIST end-effector orientation around the Y axis [rad]
og	WRIST end-effector orientation around the Z axis [rad]
enc0	**[out]** WRIST encoder reading on axis 0
enc1	**[out]** WRIST encoder reading on axis 1
enc2	**[out]** WRIST encoder reading on axis 2
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS
This feature only applies to the following devices:

- DHD_DEVICE_DELTA6
- DHD_DEVICE_OMEGA33
- DHD_DEVICE_OMEGA33_LEFT
- DHD_DEVICE_OMEGA331
- DHD_DEVICE_OMEGA331_LEFT
- DHD_DEVICE_SIGMA331
- DHD_DEVICE_SIGMA331_LEFT
- DHD_DEVICE_SIGMA33P

- DHD_DEVICE_SIGMA33P_LEFT

Returns

0 on success, -1 otherwise.
See error management for details.

4.1.4.151 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 routine 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

tx	torque on the WRIST end-effector around the X axis [Nm]
ty	torque on the WRIST end-effector around the Y axis [Nm]
tz	torque on the WRIST end-effector around the Z axis [Nm]
enc0	WRIST encoder reading on axis 0
enc1	WRIST encoder reading on axis 1
enc2	WRIST encoder reading on axis 2
mot0	**[out]** motor command around WRIST axis 0
mot1	**[out]** motor command around WRIST axis 1
mot2	**[out]** motor command around WRIST axis 2
ID	**[default=-1]** device ID (see multiple devices section for details)

Note

expert mode only - USE AT YOUR OWN RISKS
This feature only applies to the following devices:

- DHD_DEVICE_DELTA6
- DHD_DEVICE_SIGMA331
- DHD_DEVICE_SIGMA331_LEFT
- DHD_DEVICE_SIGMA33P
- DHD_DEVICE_SIGMA33P_LEFT

Returns

0 or DHD_MOTOR_SATURATED on success, -1 otherwise.
See error management for details.

5 Example Documentation

5.1 hello_world.cpp

The "hello world" DHD application.

```
#include <stdio.h>
#include "dhdc.h"


#define K   1000.0


// simple spring model which pulls the device
// towards the center of the workspace;
// if the user lifts the device 5cm above the center,
// the application exits
int
compute_my_forces (double px,  double py,  double pz,
                   double *fx, double *fy, double *fz)
{
  // spring model
  *fx = -K * px;
  *fy = -K * py;
  *fz = -K * pz;

  // exit condition
  if (pz > 0.05) return 1;
  else           return 0;
}



int
main (int  argc,
      char **argv)
{
  int    done = 0;
  double px, py, pz;
  double fx, fy, fz;

  if (dhdOpen () < 0) {
    printf ("error: cannot open device\n");
  }

  printf ("spring model applied...\n");

  while (!done) {

    // get end-effector position
    dhdGetPosition (&px, &py, &pz);

    // compute force model
    done = compute_my_forces (px, py, pz, &fx, &fy, &fz);

    // apply forces
    dhdSetForce (fx, fy, fz);

    // exit if the button is pushed
    done += dhdGetButton (0);
  }

  printf ("exiting application\n");

  dhdClose ();

  return 0;
}
```

5.2 multiple_devices.cpp

Simple example of multiple-devices programming.

```
#include <stdio.h>
#include "dhdc.h"



int
main (int  argc,
      char **argv)
{
  int done = 0;
  int deviceCount;
  int deviceID0;
```

```
  int deviceID1;

  // get device count
  deviceCount = dhdGetDeviceCount ();
  if (deviceCount < 0) {
    printf ("error: %s\n", dhdErrorGetLastStr ());
    return 0;
  }
  else if (deviceCount < 1) {
    printf ("error: no device detected\n");
  }
  else if (deviceCount < 2) {
    printf ("error: single device detected\n");
  }

  // open the first available device
  if ((deviceID0 = dhdOpenID (0)) < 0) {
    printf ("error: %s\n", dhdErrorGetLastStr ());
  }

  // open the second available device
  if ((deviceID1 = dhdOpenID (1)) < 0) {
    printf ("error: %s\n", dhdErrorGetLastStr ());
  }


  // haptic loop
  while (!done) {

    // apply a null force to put the first device in gravity compensation
    if (dhdSetForce (0.0, 0.0, 0.0, deviceID0) < 0) {
      printf ("error: %s\n", dhdErrorGetLastStr ());
      done = 1;
    }

    // apply a null force to put the second device in gravity compensation
    if (dhdSetForce (0.0, 0.0, 0.0, deviceID1) < 0) {
      printf ("error: %s\n", dhdErrorGetLastStr ());
      done = 1;
    }

    // detect button click to quit the haptic loop
    if (dhdGetButton (0)) {
      printf ("exiting...\n");
    }
  }

  // close the connection to the first device
  if (dhdClose (0) < 0) {
    printf ("error: %s\n", dhdErrorGetLastStr ());
  }

  // close the connection to the second device
  if (dhdClose (1) < 0) {
    printf ("error: %s\n", dhdErrorGetLastStr ());
  }

  return 0;
}
```

5.3 single_device.cpp

Simple example of single-device programming.

```
#include <stdio.h>
#include "dhdc.h"



int
main (int  argc,
      char **argv)
{
  int done = 0;

  // get device count
  if (dhdGetDeviceCount () <= 0) {
    printf ("error: no device found (%s)\n", dhdErrorGetLastStr ());
    return 0;
  }

  // open the first available device
  if (dhdOpen () < 0) {
    printf ("error: %s\n", dhdErrorGetLastStr ());
  }

  // haptic loop
```

```
  while (!done) {

    // apply a null force to put the device in gravity compensation
    if (dhdSetForce (0.0, 0.0, 0.0) < 0) {
      printf ("error: %s\n", dhdErrorGetLastStr ());
      done = 1;
    }

    // detect button click to quit the haptic loop
    if (dhdGetButton (0)) {
      printf ("exiting...\n");
      done = 1;
    }
  }

  // close the connection to the device
  if (dhdClose () < 0) {
    printf ("error: %s\n", dhdErrorGetLastStr ());
  }

  return 0;
}
```

Index