Haptic Device Abstraction Layer (HDAL)

API Reference

VERSION 3.0.27
March 30, 2010

Novint Technologies Incorporated
Albuquerque, NM USA

Copyright Notice

Trademarks

Warranties and Disclaimers

Novint Technologies does not warrant that this publication is error free. This publication could include technical or typographical errors or other inaccuracies. Novint may make changes to the product described in this publication or to this publication at any time, without notice.

Questions or Comments

If you have any questions for our technical support staff, please contact us at support@novint.com. You can also phone 1-866-298-4420.

If you have any questions or comments about the documentation, please contact us at support@novint.com.

Corporate Headquarters

Novint Technologies, Inc.
PO Box 66956
Albuquerque, NM 87193
Phone: 1-866-298-4420
E-mail: support@novint.com
Internet: http://www.novint.com

Preface

This manual is a reference to the Haptic Device Abstraction Layer produced by Novint Technologies. It contains reference pages to all the HDAL API functions, constants, and types. This manual was current as of the release of the corresponding version of HDAL.

The technical content of this document was mechanically generated from HDAL source code by Doxygen, a general-purpose utility for documenting C and C++ code. For more information on Doxygen, see http://www.doxygen.org.

.

Table of Contents

Overview

Files

The HDAL API is represented in two files. include\hdl\hdl.h is the primary interface to HDAL's functionality. It is complete, in that entire applications can be built from it. include\hdlu\hdlu.h is a utility interface, presenting functions that may be useful to the developer. HDAL can be used effectively without these utility interface functions.

Units

The units of measure for the HDAL interface are:

Distance	meters
Force	newtons
Time	seconds

The nominal cycle time is approximately one millisecond. However, since operating systems are not able to control intervals with precision adequate to many applications, a more precise time measure may be needed. For precise time calculations, the application should use some precision clock, such as the Windows QueryPerformanceCounter function.

Coordinate Frame

The coordinate system used by HDAL is a right hand coordinate system:

X	increases to the right
Y	increases upward
Z	increases toward the user.

The origin (X = 0, Y = 0, Z = 0) is approximately at the center of the device workspace.

HDAL Reference

C:/novint/projects/DeviceSupport/MainStream/hdal/include/hdl/hdl.h File Reference

Main API for HDAL services.

```
#include <hdl/hdlExports.h>
#include <hdl/hdlErrors.h>
#include <hdl/hdlConstants.h>
```

Defines

- #define false 0
- *logical false value for C programs* #define HDAL_ISREADY 0
- *Normal hdlGetStatus return code.* #define HDAL_NOT_CALIBRATED 0x04
- *hdlGetStatus code indicating motors not homed* #define HDAL_SERVO_NOT_STARTED 0x02
- *hdlGetStatus code indicating servo loop net yet started* #define HDAL_UNINITIALIZED 0x01
- *hdlGetStatus code indicating HDAL not yet initialized* #define HDL_BUTTON_1 0x00000001
- *Mask for button 1.* #define HDL_BUTTON_2 0x00000002
- *Mask for button 2.* #define HDL_BUTTON_3 0x00000004
- *Mask for button 3.* #define HDL_BUTTON_4 0x00000008
- *Mask for button 4.* #define HDL_BUTTON_ANY 0xffffffff
- *Mask for any button.* #define HDL_DEFAULT_DEVICE_ID 0
- *ID for the default haptic device (usually one installed).* #define HDL_INVALID_HANDLE -1
- *Handle indicating invalid device handle.* #define HDL_SERVOOP_CONTINUE 1
- *Return code for continuing servo loop.* #define HDL_SERVOOP_EXIT 0
- *Return code for exiting servo loop.* #define true 1

logical true value for C programs Typedefs

- typedef unsigned char bool
- *define bool type and values for C programmers to use with certain functions* typedef int HDLDeviceHandle
- *Handle to differentiate between multiple installed devices.* typedef int HDLDeviceID
- *ID to differentiate between multiple installed devices.* typedef int HDLOpHandle
- *Type for Servo loop operation handle.* typedef HDLServoOpExitCode __cdecl HDLServoOp (void *pParam)
- *Prototype for Servo operation function.* typedef int HDLServoOpExitCode

Type for Servo loop operation exit code. Functions

- HDLAPI __int64 HDLAPIENTRY HDL_BUILD_VERSION (HDL_VERSION_INFO_TYPE versionInfo)
- *Return Build component of version struct.* HDLAPI int HDLAPIENTRY HDL_MAJOR_VERSION (HDL_VERSION_INFO_TYPE versionInfo)
- *Return Major component of version struct.* HDLAPI int HDLAPIENTRY HDL_MINOR_VERSION (HDL_VERSION_INFO_TYPE versionInfo)
- *Return Minor component of version struct.* HDLAPI int HDLAPIENTRY hdlCatalogDevices (HDL_AVAILABILITY availability, char *serList, char **masks)
- *Catalog devices.* HDLAPI int HDLAPIENTRY hdlCountDevices ()
- *Count available devices.* HDLAPI HDLOpHandle HDLAPIENTRY hdlCreateServoOp (HDLServoOp pServoOp, void *pParam, bool bBlocking)

- *Schedule an operation (callback) to run in the servo loop.* HDLAPI void HDLAPIENTRY hdlDestroyServoOp (HDLOpHandle hServoOp)
- *Remove an operation (callback) from the servo loop.* HDLAPI bool HDLAPIENTRY hdlDeviceDisconnected ()
- *Return connection state of device.* HDLAPI const char *HDLAPIENTRY hdlDeviceModel ()
- *Return the device model string.* HDLAPI void HDLAPIENTRY hdlDeviceWorkspace (double workspaceDimensions[6])
- *Retrieve the workspace of the device, measured in meters.* HDLAPI HDLError HDLAPIENTRY hdlGetError ()
- *Return the current error code from the error stack.* HDLAPI unsigned int HDLAPIENTRY hdlGetState ()
- *Query HDAL state.* HDLAPI bool HDLAPIENTRY hdlGetVersion (HDL_VERSION_REQUEST requestType, HDL_VERSION_INFO_TYPE *versionInfo)
- *Get version information.* HDLAPI void HDLAPIENTRY hdlGripGetAttribute (HDL_GRIP_ATTRIBUTE attributeType, int index, int *value)
- *Get grip attribute as an int.* HDLAPI void HDLAPIENTRY hdlGripGetAttributeb (HDL_GRIP_ATTRIBUTE attributeType, int index, bool *value)
- *Get grip attribute as a bool.* HDLAPI void HDLAPIENTRY hdlGripGetAttributed (HDL_GRIP_ATTRIBUTE attributeType, int index, double *value)
- *Get grip attribute as a double.* HDLAPI void HDLAPIENTRY hdlGripGetAttributes (HDL_GRIP_ATTRIBUTE attributeType, int count, int *value)
- *Get grip attribute as an array of int.* HDLAPI void HDLAPIENTRY hdlGripGetAttributesb (HDL_GRIP_ATTRIBUTE attributeType, int count, bool *value)
- *Get grip attribute as a array of bool.* HDLAPI void HDLAPIENTRY hdlGripGetAttributesd (HDL_GRIP_ATTRIBUTE attributeType, int count, double *value)
- *Get grip attribute as a array of double.* HDLAPI void HDLAPIENTRY hdlGripGetAttributesv (HDL_GRIP_ATTRIBUTE attributeType, int count, double *value)
- *Get grip attribute as a array of vector.* HDLAPI void HDLAPIENTRY hdlGripGetAttributev (HDL_GRIP_ATTRIBUTE attributeType, int index, double *value)
- *Get grip attribute as a vector.* HDLAPI void HDLAPIENTRY hdlGripQuery (HDL_GRIP_ATTRIBUTE attributeID, int *num)
- *Retrieve the number of attributes of a given type on the attached grip.* HDLAPI void HDLAPIENTRY hdlGripSetAttribute (HDL_GRIP_ATTRIBUTE attributeType, int index, int *value)
- *Get grip attribute as an int.* HDLAPI void HDLAPIENTRY hdlGripSetAttributeb (HDL_GRIP_ATTRIBUTE attributeType, int index, bool *value)
- *Get grip attribute as a bool.* HDLAPI void HDLAPIENTRY hdlGripSetAttributed (HDL_GRIP_ATTRIBUTE attributeType, int index, double *value)
- *Get grip attribute as a double.* HDLAPI void HDLAPIENTRY hdlGripSetAttributes (HDL_GRIP_ATTRIBUTE attributeType, int count, int *value)
- *Get grip attribute as an array of int.* HDLAPI void HDLAPIENTRY hdlGripSetAttributesb (HDL_GRIP_ATTRIBUTE attributeType, int count, bool *value)
- *Get grip attribute as a array of bool.* HDLAPI void HDLAPIENTRY hdlGripSetAttributesd (HDL_GRIP_ATTRIBUTE attributeType, int count, double *value)
- *Get grip attribute as a array of double.* HDLAPI void HDLAPIENTRY hdlGripSetAttributesv (HDL_GRIP_ATTRIBUTE attributeType, int count, double *value)
- *Get grip attribute as a array of vector.* HDLAPI void HDLAPIENTRY hdlGripSetAttributev (HDL_GRIP_ATTRIBUTE attributeType, int index, double *value)
- *Get grip attribute as a vector.* HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitDevice (HDLDeviceID deviceID)
- *Initialize a haptic device.* HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitDeviceBySerialNumber (const char *serNum, const char *configPath)
- *Initialize a specific haptic device by its serial number.* HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitIndexedDevice (const int index, const char *configPath)

- *Initialize a specific indexed haptic device.* HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitIndexedDeviceEx (const int index, const char *configPath, HDLServoOp *callBack, void *pData)
- *Extended-initialize a specific indexed haptic device.* HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitNamedDevice (const char *deviceName, const char *configPath)
- *Initialize a specific named haptic device.* HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitNamedDeviceEx (const char *deviceName, const char *configPath, HDLServoOp *callBack, void *pData)
- *Extended-initialize a specific named haptic device.* HDLAPI void HDLAPIENTRY hdlMakeCurrent (HDLDeviceHandle hHandle)
- *Make a specific haptic device current (Allows application to send forces to a specific device).* HDLAPI void HDLAPIENTRY hdlSetToolForce (double force[3])
- *Set the force to be generated by the device, measured in newtons.* HDLAPI void HDLAPIENTRY hdlStart ()
- *Start servo and all haptic devices.* HDLAPI void HDLAPIENTRY hdlStop ()
- *Stop servo and all haptic devices.* HDLAPI void HDLAPIENTRY hdlToolButton (bool *pButton)
- *Return current state of tool button(s).* HDLAPI void HDLAPIENTRY hdlToolButtons (int *pButton)
- *Return current state of tool buttons.* HDLAPI void HDLAPIENTRY hdlToolPosition (double position[3])
- *Return current tool position.* HDLAPI void HDLAPIENTRY hdlUninitDevice (HDLDeviceHandle hHandle)

Uninitializes a haptic device.

Detailed Description

Main API for HDAL services.

Haptic Device Abstraction Layer Low level, cross-platform, general purpose interface.

Definition in file hdl.h.

Typedef Documentation

typedef int HDLDeviceHandle

Handle to differentiate between multiple installed devices.

Handle is an abstraction returned by the initialization routine.

Definition at line 46 of file hdl.h.

typedef HDLServoOpExitCode __cdecl HDLServoOp(void *pParam)

Prototype for Servo operation function.

Parameters:

[out] *pParam* Pointer to data required by operation

Returns:

Exit code

Errors: None

Definition at line 77 of file hdl.h.

Function Documentation

HDLAPI __int64 HDLAPIENTRY HDL_BUILD_VERSION (HDL_VERSION_INFO_TYPE *versionInfo*)

Return Build component of version struct.

Parameters:

[in] *versionInfo* HDL_VERSION_INFO_TYPE struct returned from hdlGetVersion()

Returns:

Build component

See also:

hdlGetVersion().
Errors: None.

HDLAPI int HDLAPIENTRY HDL_MAJOR_VERSION (HDL_VERSION_INFO_TYPE *versionInfo*)

Return Major component of version struct.

Parameters:

[in] *versionInfo* HDL_VERSION_INFO_TYPE struct returned from hdlGetVersion()

Returns:

Major component

See also:

hdlGetVersion(). Errors: None.

HDLAPI int HDLAPIENTRY HDL_MINOR_VERSION (HDL_VERSION_INFO_TYPE *versionInfo*)

Return Minor component of version struct.

Parameters:

[in] *versionInfo* HDL_VERSION_INFO_TYPE struct returned from hdlGetVersion()

Returns:

Minor component

See also:

hdlGetVersion().
Errors: None.

HDLAPI int HDLAPIENTRY hdlCatalogDevices (HDL_AVAILABILITY *availability*, char * *serList*, char ** *masks*)

Catalog devices.

Parameters:

[in] *availability* Selector describing which devices are "available"
[out] *serList* Pointer to receiver of list of serial numbers (or NULL)
[in] *masks* Pointer to list of masks for hiding devices

Returns:

Number of available devices not masked by "masks" parameter
Errors: none

Note:

"masks" is a null-terminated list of pointers to strings of eight characters each. If a device's serial number "matches" a string, it is said to be "masked" and will not appear in the returned list. For a character to be masked, each character in the serial number must "match" the corresponding

character in the mask. Characters match under the following rules: Anything matches a '.' in the mask Alphabetic characters match a '@' in the mask Digits match a '*' in the mask For any other character in the mask, only the same character matches. If the masks parameter is NULL, {"@.....", NULL} is used, to correspond to Novint's usage for "auxiliary" devices. .
The list is a simple character array of n*9 characters, where n is the maximum number of devices expected. Each serial number starts every 9 characters and is a proper null-terminated C string. Typical usage is to call it once with NULL for the list parameter, use the result to allocate space for that many 9-byte arrays (8 characters plus the null terminator), then call it again with a pointer to that storage location.
See the note about calling frequency at hdlCountDevices.

HDLAPI int HDLAPIENTRY hdlCountDevices ()

Count available devices.

Parameters:

None

Returns:

Number of available devices
Errors: None

Note:

Valid for Novint Falcon devices only. A device is considered "available" if it is connected and not already opened for communication by any application. For instance, if two devices are connected, before hdlInit****Device() is called on either of them, this function will return 2. After the initing one device, this function will return 1. Internally, this function calls hdlCatalogDevices with masks = {"@.....", NULL}. See the notes on hdlCatalogDevices for a full explanation, but in brief, this means that hdlCountDevices does not count Novint's "auxiliary" devices.
DO NOT call this function within a synchronization callback or as part of the normal servo callback. It is such a time-consuming operation that it will degrade performance unacceptably. For example, with two Falcons connected, the enumeration time on a dual-core 2GHz XP machine required about 80 msec. Rather, it should be called infrequently, such as normal application state change opportunities.

HDLAPI HDLOpHandle HDLAPIENTRY hdlCreateServoOp (HDLServoOp *pServoOp*, void * *pParam*, bool *bBlocking*)

Schedule an operation (callback) to run in the servo loop.

Operation is either blocking (client waits until completion) or non-blocking (client continues execution).

Parameters:

[in] *pServoOp* Pointer to servo operation function
[in] *pParam* Pointer to data for servo operation function
[in] *bBlocking* Flag to indicate whether servo loop blocks

Returns:

Handle to servo operation entry
Errors: None

See also:

hdlDestroyServoOp, hdlInitNamedDevice, hdlInitIndexedDevice

HDLAPI void HDLAPIENTRY hdlDestroyServoOp (HDLOpHandle *hServoOp*)

Remove an operation (callback) from the servo loop.

Parameters:

[in] *hServoOp* Handle to servo op to remove

Returns:

Nothing
Errors: None.

Note:

hdlDestroyServoOp() should be called at application termination time for any servo operation that was added with bBlocking = false.

See also:

hdlCreateServoOp, hdlInitNamedDevice, hdlIndexedDevice

HDLAPI const char* HDLAPIENTRY hdlDeviceModel ()

Return the device model string.

Parameters:

None

Returns:

Device model string
Errors: None

HDLAPI void HDLAPIENTRY hdlDeviceWorkspace (double *workspaceDimensions*[6])

Retrieve the workspace of the device, measured in meters.

Call this function to retrieve the workspace of the current device. Since not all devices have the same physical workspace dimensions, the application must account for different device workspaces. The workspace is defined in the device reference coordinate frame. It is up to the user to transform positions in this coordinate frame into the application's coordinate frame. See hdluGenerateHapticToAppWorkspaceTransform for a utility function to assist in this.

Dimension order: minx, miny, minz, maxx, maxy, maxy (left, bottom, far, right, top, near)

(minx, miny, minz) are the coordinates of the left-bottom-far corner of the device workspace.

(maxx, maxy, maxz) are the coordinates of the right-top-near corner of the device workspace.

Parameters:

[out] *workspaceDimensions* See explanation above.

Returns:

Nothing
Errors:

- manufacturer specific
- no current device

HDLAPI HDLError HDLAPIENTRY hdlGetError ()

Return the current error code from the error stack.

Returns:

Error code on the top of the error stack.
HDL_NO_ERROR if the error stack is empty.

HDLAPI unsigned int HDLAPIENTRY hdlGetState ()

Query HDAL state.

Parameters:

None

Returns:

Nothing

Errors: manufacturer specific

Note:

If return == HDAL_ISREADY, device is ready. Otherwise, test to see reason:
return && XXXX != 0, where XXXX is
HDAL_UNINITIALIZED hdlInitNamedDevice failed earlier
HDAL_SERVO_NOT_STARTED hdlStart() not called earlier
HDAL_NOT_CALIBRATED needs autocalibration

HDLAPI bool HDLAPIENTRY hdlGetVersion (HDL_VERSION_REQUEST *requestType*, HDL_VERSION_INFO_TYPE * *versionInfo*)

Get version information.

Parameters:

[in] *requestType* Type of version information requested
[out] *versionInfo* Requested info

Returns:

Success or failure
Errors: None.
Typical usage:

```
HDL_VERSION_INFO_TYPE deviceVersion;
int                   deviceMajor;
int                   deviceMinor;
__int64               deviceSerialNumber;
if (hdlGetVersion(HDL_DEVICE, &deviceVersion))
{
    deviceMajor = HDL_MAJOR_VERSION(deviceVersion);
    deviceMinor = HDL_MINOR_VERSION(deviceVersion);
    deviceSerialNumber = HDL_BUILD_VERSION(deviceVersion);
}
```

HDLAPI void HDLAPIENTRY hdlGripQuery (HDL_GRIP_ATTRIBUTE *attributeID*, int * *num*)

Retrieve the number of attributes of a given type on the attached grip.

Parameters:

[in] *attributeID* ID of attribute of interest
[out] *num* Count of attributes of requested type
Errors: None

Note:

Returns 0 in num parameter if no components of the type exist.

HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitDevice (HDLDeviceID *deviceID*)

Initialize a haptic device.

Parameters:

[in] *deviceID* ID of haptic device

Returns:

Handle to haptic device
Errors:

- manufacturer specific
- could not load device specific dll

Deprecated:

Only supports a single device, deviceID is ignored. Included to support older apps. Use hdlInitNamedDevice instead.

See also:

hdlInitNamedDevice

HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitDeviceBySerialNumber (const char * *serNum*, const char * *configPath*)

Initialize a specific haptic device by its serial number.

Parameters:

[in] *serNum* Serial number of haptic device.
[in] *configPath* Path/file for ini file.
configPath search order:

See also:

hdlInitNamedDevice

Returns:

Handle to haptic device
Errors:

- manufacturer specific
- could not load device specific dll

Note:

- Support only Falcon devices via serial number

Setup sequence:

```
char SerialNumbers[5][9];  // space for 5 devices; 8-char serial numbers
int m = hdlCatalogDevices(HDL_ALL_DEVICES, SerialNumbers, 0);
char* serNum = < search for desired device >
hdlInitDeviceBySerialNumber(serNum); // 0 <= n < m
hdlStart();
hdlCreateServoOp(...);
```

Teardown sequence:

```
hdlDestroyServoOp(...);
hdlStop();
hdlUninitDevice(...);
```

See also:

hdlInitNamedDevice, hdlStart, hdlStop, hdlCreateServoOp, hdlDestroyServoOp, hdlUninitDevice

Note:

In C++ programs, configPath is optional, with a default value of (const char *) 0.
C programs must pass (const char *) 0 to use default configPath

HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitIndexedDevice (const int *index*, const char * *configPath*)

Initialize a specific indexed haptic device.

Parameters:

[in] *index* Index of haptic device.
[in] *configPath* Path/file for ini file.
configPath search order:

See also:

hdlInitNamedDevice

Returns:

Handle to haptic device
Errors:

- manufacturer specific
- could not load device specific dll

Note:

- Support only Falcon devices via index
- Index refers to alphabetical sort order by serial number

Setup sequence:

```
m = hdlCountDevices();
hdlInitIndexedDevice(n); // 0 <= n < m
hdlStart();
hdlCreateServoOp(...);
```

Teardown sequence:

```
hdlDestroyServoOp(...);
hdlStop();
hdlUninitDevice(...);
```

See also:

hdlInitNamedDevice, hdlStart, hdlStop, hdlCreateServoOp, hdlDestroyServoOp, hdlUninitDevice

Note:

In C++ programs, configPath is optional, with a default value of (const char *) 0.
C programs must pass (const char *) 0 to use default configPath

HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitIndexedDeviceEx (const int *index*, const char * *configPath*, HDLServoOp * *callBack*, void * *pData*)

Extended-initialize a specific indexed haptic device.

Parameters:

[in] *index* Index of haptic device.
[in] *configPath* Path/file for ini file.
[in] *callBack* Callback function to signal interruption
[in] *pData* Pointer to data structure passed to callback function

Note:

- Like hdlInitNamedDevice except for additional pointer to callback function. If not null, another init will cause this one to be closed and the callback function to be called. In general, this will be used to signal the original caller that its session with the Falcon has been interrupted
- Usage

```
hdlInitIndexedDevice(...interruptCB);
hdlStart();
hdlCreateServoOp(...);
```

- Teardown sequence:

```
hdlDestroyServoOp(...);
hdlStop();
hdlUninitDevice(...);
...
// This function will be called from the original servo thread as the
//   last act before stopping the servo and releasing the device.  The
//   user must take appropriate action inside the client process.
HDLServoOpExitCode interruptCB(void* pUserData)
{
    interrupted = true;
    //... other action as appropriate
}
```

-

See also:

hdlInitNamedDevice

HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitNamedDevice (const char * *deviceName*, const char * *configPath*)

Initialize a specific named haptic device.

Parameters:

[in] *deviceName* Name of haptic device.
[in] *configPath* Path/file for ini file.
configPath search order:

1. relative to executable directory
2. relative to executable directory's parent if executable directory is Debug or Release
3. config directory in path specified by NOVINT_DEVICE_SUPPORT

Returns:

Handle to haptic device
Errors:

- manufacturer specific
- could not load device specific dll

Note:

- Support multiple devices via deviceName string.

Setup sequence:

```
hdlInitNamedDevice(...);
hdlStart();
hdlCreateServoOp(...);
```

Teardown sequence:

```
hdlDestroyServoOp(...);
hdlStop();
hdlUninitDevice(...);
```

See also:

hdlInitIndexedDevice, hdlStart, hdlStop, hdlCreateServoOp, hdlDestroyServoOp, hdlUninitDevice

Note:

In C++ programs, configPath is optional, with a default value of (const char *) 0.
C programs must pass (const char *) 0 to use default configPath

HDLAPI HDLDeviceHandle HDLAPIENTRY hdlInitNamedDeviceEx (const char * *deviceName*, const char * *configPath*, HDLServoOp * *callBack*, void * *pData*)

Extended-initialize a specific named haptic device.

Parameters:

[in] *deviceName* Name of haptic device.
[in] *configPath* Path/file for ini file.
[in] *callBack* Callback function to signal interruption
[in] *pData* Pointer to data structure passed to callback function

Note:

- Like hdlInitNamedDevice except for additional pointer to callback function. If not null, another init will cause this one to be closed and the callback function to be called. In general, this will be used to signal the original caller that its session with the Falcon has been interrupted
- Usage

```
hdlInitNamedDevice(...interruptCB);
hdlStart();
hdlCreateServoOp(...);
```

- Teardown sequence:

```
hdlDestroyServoOp(...);
hdlStop();
hdlUninitDevice(...);
...
// This function will be called from the original servo thread as the
//   last act before stopping the servo and releasing the device.  The
//   user must take appropriate action inside the client process.
HDLServoOpExitCode interruptCB(void* pUserData)
```

```
{
    interrupted = true;
    //... other action as appropriate
}
```

-

See also:

hdlInitNamedDevice

HDLAPI void HDLAPIENTRY hdlMakeCurrent (HDLDeviceHandle *hHandle*)

Make a specific haptic device current (Allows application to send forces to a specific device).

Parameters:

[in] *hHandle* Haptic device handle

Returns:

Nothing

Errors:

- manufacturer specific
- hHandle invalid

HDLAPI void HDLAPIENTRY hdlSetToolForce (double *force*[3])

Set the force to be generated by the device, measured in newtons.

Forces are in device coordinates. Dimension order: x, y, z

Parameters:

[in] *force* Measured in Newtons; x, y, z order

Returns:

Nothing

Errors:

- manufacturer specific
- no current device
- max force exceeded

HDLAPI void HDLAPIENTRY hdlStart ()

Start servo and all haptic devices.

Parameters:

None

Returns:

Handle to haptic device

Errors:

- manufacturer specific
- servo could not start

Note:

Starts servo and all haptic devices. Call after all devices are initialized. Start is separated from hdlInitNamedDevice to allow all devices to be initialized before servo operations are started. Currently, only one Falcon at a time is supported, but this restriction will be lifted in the future. Other device types supported by HDAL may already allow multiple devices to be connected.

See also:

hdlStop, hdlInitNamedDevice, hdlInitIndexedDevice

HDLAPI void HDLAPIENTRY hdlStop ()

Stop servo and all haptic devices.

In its current form, hdlStop() does nothing. In the future, it will at least stop processing callback functions. Other changes in behavior are yet to be defined and will be described when implemented.

Parameters:

None

Returns:

Nothing

See also:

hdlStart, hdlInitNamedDevice, hdlInitIndexedDevice

HDLAPI void HDLAPIENTRY hdlToolButton (bool * *pButton*)

Return current state of tool button(s).

For multi-button devices, if any button is pressed, pButton* is set to true.

Parameters:

[out] *pButton* Pointer to bool to hold button state

Returns:

Nothing
Errors: None

HDLAPI void HDLAPIENTRY hdlToolButtons (int * *pButton*)

Return current state of tool buttons.

Returned value is a bitmask of buttons, with the least significant bit associated with button "0".

Parameters:

[out] *pButton* Pointer to an int to hold button states

Returns:

Nothing
Errors: None

HDLAPI void HDLAPIENTRY hdlToolPosition (double *position*[3])

Return current tool position.

Parameters:

[out] *position* In x, y, z order, measured in meters.

Returns:

Nothing
Errors: None

HDLAPI void HDLAPIENTRY hdlUninitDevice (HDLDeviceHandle *hHandle*)

Uninitializes a haptic device.

Parameters:

[in] *hHandle* Handle of haptic device

Returns:

Nothing
Errors: manufacturer specific

See also:
hdlInitNamedDevice, hdlInitIndexedDevice

C:/novint/projects/DeviceSupport/MainStream/hdal/include/hdl/hdlConstants.h File Reference

Constants for HDAL.

Defines

- #define HDL_MAX_DEVICES 6
- *Maximum device count.* #define HDL_SERNUM_BUFFSIZE 9
- *Length of buffer for serial number (includes null terminator).* #define HDL_VERSION_INVALID -1
- *version is invalid* #define HDL_VERSION_NOT_APPLICABLE -2
- *version field not applicable* #define HDL_VERSION_UNAVAILABLE -3

version field not available Typedefs

- typedef __int64 HDL_VERSION_INFO_TYPE

Structure returned by hdlGetVersion. Enumerations

- enum HDL_AVAILABILITY { **HDL_ALL_DEVICES**, **HDL_NOT_OPEN_BY_OTHER_APP**, **HDL_NOT_OPEN_BY_ANY_APP**, **HDL_DEFAULT_AVAILABILITY** }
  Device availability control.
- enum HDL_GRIP_ATTRIBUTE { HDL_GRIP_ID, HDL_GRIP_HELD, HDL_GRIP_POSITION, HDL_GRIP_ORIENTATION, HDL_GRIP_QUATERNION, HDL_GRIP_AXISANGLE, HDL_GRIP_OFFSET, HDL_GRIP_ANGLE, HDL_GRIP_FORCE, HDL_GRIP_BUTTON, HDL_GRIP_FLOAT, HDL_GRIP_TORQUE, HDL_GRIP_RGB_LIGHT, **HDL_GRIP_RED_LIGHT**, **HDL_GRIP_GREEN_LIGHT**, **HDL_GRIP_BLUE_LIGHT**, HDL_GRIP_MEMORY_ROM, HDL_GRIP_MEMORY_RAM, HDL_GRIP_MASS, HDL_GRIP_GRAVITY_COMP, HDL_GRIP_RAW_ENCODER, HDL_GRIP_RAW_MOTOR }
  Grip Attributes.
- enum HDL_VERSION_REQUEST { HDL_HDAL = 0x11, HDL_DEVICE = 0x21, HDL_DEVICE_SDK = 0x22, HDL_DEVICE_COMMS = 0x23, HDL_DEVICE_OS = 0x24, HDL_GRIP = 0x33 }
  Enumeration of version request types.

Detailed Description

Constants for HDAL.

Definition in file hdlConstants.h.

Typedef Documentation

typedef __int64 HDL_VERSION_INFO_TYPE

Structure returned by hdlGetVersion.

See also:
hdlGetVersion

Definition at line 39 of file hdlConstants.h.

Enumeration Type Documentation

enum HDL_GRIP_ATTRIBUTE

Grip Attributes.

Enumerator:

HDL_GRIP_ID ID of the attached grip.

HDL_GRIP_HELD is grip being held

HDL_GRIP_POSITION position of grip "center"

HDL_GRIP_ORIENTATION orientation of grip

HDL_GRIP_QUATERNION grip orientation as a quaternion

HDL_GRIP_AXISANGLE grip orientation as an axis-angle pair

HDL_GRIP_OFFSET offset from Falcon base center (center of face plate)

HDL_GRIP_ANGLE Return all four raw angles from a 4DOF grip.

HDL_GRIP_FORCE force sent to grip

HDL_GRIP_BUTTON grip button

HDL_GRIP_FLOAT grip slider/knob

HDL_GRIP_TORQUE torque sent to grip

HDL_GRIP_RGB_LIGHT Badge LEDs.

HDL_GRIP_MEMORY_ROM return the contents of the attached grip ROM

HDL_GRIP_MEMORY_RAM return the contents of the attach grip RAM

HDL_GRIP_MASS return the grip mass, in kg

HDL_GRIP_GRAVITY_COMP set or return the state of gravity compensation

HDL_GRIP_RAW_ENCODER return raw 4DOF encoder values

HDL_GRIP_RAW_MOTOR return most recent raw motor commands

Definition at line 43 of file hdlConstants.h.

enum HDL_VERSION_REQUEST

Enumeration of version request types.

See also:

hdlGetVersion

Enumerator:

HDL_HDAL version of HDAL

HDL_DEVICE device hardware in current device context

HDL_DEVICE_SDK SDK version of current device.

HDL_DEVICE_COMMS communications version of current device

HDL_DEVICE_OS version of device OS

HDL_GRIP grip in current device context

Definition at line 22 of file hdlConstants.h.

C:/novint/projects/DeviceSupport/MainStream/hdal/include/hdl/hdlErrors.h File Reference

Error codes returned from HDAL.

Defines

- #define HDL_ERROR_INIT_FAILED 0x10
- *Device initialization error.* #define HDL_ERROR_INTERNAL 0x02
- *HDAL internal error>.* #define HDL_ERROR_STACK_OVERFLOW 0x01
- *Overflow of error stack.* #define HDL_INIT_DEVICE_ALREADY_INITED 0x16
- *Device already initialized.* #define HDL_INIT_DEVICE_FAILURE 0x15
- *Failed to initilize device.* #define HDL_INIT_DEVICE_NOT_CONNECTED 0x17
- *Requested device not connected.* #define HDL_INIT_DLL_LOAD_ERROR 0x14
- *Could not load driver DLL.* #define HDL_INIT_ERROR_MASK 0x1F
- *Mask for all initialization errors.* #define HDL_INIT_INI_DLL_STRING_NOT_FOUND 0x12
- *No DLL name in configuration file.* #define HDL_INIT_INI_MANUFACTURER_NAME_STRING_NOT_FOUND 0x13
- *No MANUFACTURER_NAME value in configuration file.* #define HDL_INIT_INI_NOT_FOUND 0x11
- *Could not find configuration file.* #define HDL_NO_ERROR 0x0
- *No errors on error stack.* #define HDL_SERVO_START_ERROR 0x18

Could not start servo thread. Typedefs

- typedef int HDLError

HDAL API Errors.

Detailed Description

Error codes returned from HDAL.

Definition in file hdlErrors.h.

Typedef Documentation

typedef int HDLError

HDAL API Errors.

Client application queries HDAL errors using hdlGetError(). hdlGetError() returns an error type.

Definition at line 19 of file hdlErrors.h.

C:/novint/projects/DeviceSupport/MainStream/hdal/include/hdlu/hdlu.h File Reference

Utility functions for HDAL applications.

```
#include <hdl/hdlExports.h>
```

Functions

- HDLAPI void HDLAPIENTRY hdluGenerateHapticToAppWorkspaceTransform (double hapticWorkspace[6], double gameWorkspace[6], bool useUniformScale, double transformMat[16])
- *Generate transform for mapping between haptic and game workspace.* HDLAPI double hdluGetSystemTime (void)

Compute a precise time based on CPU high performance timer.

Detailed Description

Utility functions for HDAL applications.

Haptic Device Abstraction Layer Low level, cross-platform, general purpose interface.

Definition in file hdlu.h.

Function Documentation

HDLAPI void HDLAPIENTRY hdluGenerateHapticToAppWorkspaceTransform (double *hapticWorkspace*[6], double *gameWorkspace*[6], bool *useUniformScale*, double *transformMat*[16])

Generate transform for mapping between haptic and game workspace.

Inputs specify the minimum and maximum coordinate values of rectangular paralleliped reference boxes. The meanings of the two boxes is arbitrary. The names "hapticWorkspace" and "gameWorkspace" derive from a common usage to transform from device (haptic workspace) units to game workspace units. In such usage, hdlDeviceWorkspace can be used to get an approximate range of values, but the developer must be aware that this range is approximate, so the resultant transform is only approximate. The function computes and returns (in transformMat) the transform matrix that will convert position data relative to the "hapticWorkspace" reference into position data relative to the gameWorkspace reference.

Parameters:

[in] *hapticWorkspace* minx, miny, minz, maxx, maxy, maxz
[in] *gameWorkspace* minx, miny, minz, maxx, maxy, maxz
[in] *useUniformScale* If true, scale the same in all three dimenstions
[out] *transformMat* Transformation from haptic to game workspace

Returns:

Nothing

Errors: None

HDLAPI double hdluGetSystemTime (void)

Compute a precise time based on CPU high performance timer.

Parameters:

None

Returns:

Current system time, in seconds from start of epoch

Index