3D TOUCH™ SDK

OPENHAPTICS™ TOOLKIT

VERSION 1.0

API REFERENCE

SensAble technologies

Questions or Comments

If you have any questions for our technical support staff, please contact us at support@sensable.com. You can also phone 1-888-SENSABL (U.S.A. only) or 1-781-937-8315 (International).

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

Corporate Headquarters

SensAble Technologies, Inc.
15 Constitution Way
Woburn, MA 01801
Phone: 1-888-SENSABL (U.S.A. only) or 1-781-937-8315 (International)
E-mail: support@sensable.com
Internet: http://www.sensable.com

Last Updated - 6 August, 2004 11:24 am P/N - 02524 R3

Preface

This manual is meant to be used as a companion to the *3D Touch Programmer's Guide*. It contains reference pages to all the OpenHaptics™ HDAPI and HLAPI functions and types as well as appendices with tables that describe all the parameters. Functions are listed alphabetically by section.

A more recent version of this document may be available for download from the SensAble online Developer Support Center (DSC). To access the DSC, visit the SensAble Support page at http://www.sensable.com/support/.

The DSC provides 24/7 access to the most current information and forums for the SensAble 3D Touch™ and GHOST® SDKs.

Please note that you will need to register for a user name and password to access the DSC.

Typographical Conventions

This guide uses the following typographical conventions:

Convention	Description	Example
Italics	First use of a new term; reference to another document or file.	See the *User Manual.*
`Courier`	Identifies code.	
Note, Warning, Important	Call out important additional information.	

Note Code snippets in this document may include hard and soft line breaks for formatting purposes.

Contents

Section I: HDAPI Routines

1

Device Routines

The following are routines for managing the device and forces. This includes all functionality for initializing devices, querying and setting device state, and turning capabilities on or off.

hdBeginFrame

Description: Begins a haptics frame, which is a block of code within which the device state is guaranteed to be consistent. Updates state from the device for current/last information. All state-related information, such as setting state, should be done within a haptics frame.

Syntax:

```
void hdBeginFrame(HHD hHD)
```

Argument	Definition
hHD	The device handle of an initialized device.

Returns: void

Usage: Typically the first haptics call per device per scheduler tick. For example, if two haptics devices are being managed, hdBeginFrame should be called for each of them before device-specific calls are made. Only one frame is allowed per device per scheduler tick unless HD_ONE_FRAME_LIMIT is disabled. However, frames for the same device can be nested within a scheduler tick. This function automatically makes the supplied device current.

Example;

```
HHD hHD = hdGetCurrentDevice;
hdBeginFrame(hHD);
```

Errors:	HD_ILLEGAL_BEGIN if a frame was already completed for the current device within the current scheduler tick.
See also:	hdMakeCurrentDevice, hdDisable, hdIsEnabled.

---

hdDisable

Description: Disables a capability.

Syntax: `void hdDisable(HDenum cap)`

Argument	Definition
cap	The capability to disable. For a list of the capabilities see “Capability Parameters” on page A-10 in Appendix A, “Haptic Device API Parameters”.

Returns: void

Usage: Capabilities are typically related to safety mechanisms. Extreme caution should be used when disabling safety features.

Example: `hdDisable(HD_MAX_FORCE_CLAMPING);`

Errors: HD_INVALID_ENUM if cap does not support enable/disable.

See Also: hdEnable, hdIsEnabled

hdDisableDevice

Description: Disables a device. The handle should not be used afterward.

Syntax: `void hdDisableDevice(HHD hHD)`

Argument	Definition
hHD	The device handle of an initialized device.

Returns: void

Usage: Call during cleanup when done using a device. Typically the last call after stopping the scheduler and unscheduling all scheduled callbacks.

Examples:

```
hdStopScheduler();
hdUnschedule(scheduleCallbackHandle);
hdDisableDevice(hdGetCurrentDevice());
```

Errors: none

See also: hdInitDevice, hdStopScheduler

hdEnable

Description: Enables a capability.

Syntax: `void hdEnable(HDenum cap)`

Argument	Definition
cap	The capability to enable. For a list of the capabilities see Table A-9, “hdEnable, hdDisable Parameters,” on page A-10.

Returns: void

Usage: Capabilities are typically related to safety mechanisms. Most are turned on by default.

Example: `hdEnable(HD_FORCE_OUTPUT);`

Errors: HD_INVALID_ENUM if cap does not support enable/disable.

See also: hdDisable, hdIsEnabled

hdEndFrame

Description: Ends a haptics frame. Causes forces and other state to be written to the device. A begin and end should always be paired within the same scheduler tick.

Syntax:

```
void hdEndFrame(HHD hHD)
```

Argument	Definition
hHD	The device handle of an initialized device.

Returns: void

Usage: Typically the last haptics call per device, per scheduler tick.

Example:

```
hdEndFrame(hdGetCurrentDevice());
```

Errors: HD_ILLEGAL_END if this call is not paired correctly with a hdBeginFrame of the same device handle.

See also: hdBeginFrame, hdMakeCurrentDevice

hdGet (Parameter Values)

Description: Obtains information about the device. There are five query functions for obtaining information about the device associated with the parameter name used.

Syntax:

```
void hdGetBooleanv(HDenum pname, HDboolean *params)
void hdGetIntegerv(HDenum pname, HDint *params)
void hdGetFloatv(HDenum pname, HDfloat *params)
void hdGetDoublev(HDenum pname, HDdouble *params)
void hdGetLongv(HDenum pname, HDlong *params)
```

Argument	Definition
pname	Parameter name to use
params	Array where the results will be returned. For a list of parameters Table A-1, “hdGet Parameters,” on page A-2.

Returns: void

Usage: Primary function for getting device information. Depending on the parameter, one should use the appropriate params type, and therefore the appropriate function signature. The caller of the function has the responsibility for allocating memory. These functions should only be called within a haptics frame.

Example:

```
HDdouble position[3];
hdGetDoublev(HD_CURRENT_POSITION,position);

HDint buttons;
hdGetIntegerv(HD_CURRENT_BUTTONS,&buttons);
```

Errors:

- HD_INVALID_INPUT_TYPE if pname does not support the input type.
- HD_INVALID_ENUM if pname does not support hdGet.

See also: hdSet (Parameter Values)

hdGetCurrentDevice

Description: Gets the handle of the current device.

Syntax:

```
HHD hdGetCurrentDevice()
```

Returns: The handle of the current device.

Usage: Primarily used in multi-device applications to keep track of which device is current, or for calls that require a device handle.

Example:

```
HHD hHD = hdGetCurrentDevice();
```

Errors: HD_INVALID_HANDLE if no device is current. For example, if no device has been initiated yet.

See also: hdMakeCurrentDevice

hdGetError

Description: Returns errors in order from most recent to least. Each call retrieves and removes one error from the error stack. If no error exists, this function returns a HDErrorInfo with HD_SUCCESS as its code. HDErrorInfo contains the error code from the defines file, the handle of the device that was active when the error occurred, and the device's original internal error code. The internal code can be used for obtaining additional support from the device vendor.

Syntax:

```
HDErrorInfo hdGetError()
```

Returns: HDErrorInfo. This data structure is a typedef in hdDefines.h. It exposes information on the handle of the device, an errorCode, and an internal error code.

Usage: Intersperse in code to occasionally check for errors.

Example:

```
HDErrorInfo error;
error = hdGetError();
if (HD_DEVICE_ERROR(error))
    // do error handling
    hdGetErrorString(error.errorCode);
```

Errors: none

See also: hdGetErrorString, HDErrorInfo Type

hdGetErrorString

Description:	Returns information about an error code.
Syntax:	`HDstring hdGetErrorString(HDerror errorCode)`

Argument	Definition
errorCode	An error code from hdDefines

Returns:	A readable string representing the explanation of the error code.
Usage:	Obtains useful information about error codes.
Example:	`hdGetErrorString(HD_FRAME_ERROR);`
Errors:	none
See also:	hdGetError, HD_DEVICE_ERROR

hdGetString

Description:	Gets a string value for the associated parameter name.
Syntax:	`HDstring hdGetString(HDenum pname)`

Argument	Definition
pname	Parameter name to use. For a list what types and how many values are used by each parameter name, see Table A-1, “hdGet Parameters,” on page A-2.

**Not all parameter names support string values.

Returns:	Requested string associated with the parameter name.
Usage:	Gets readable string information about device properties, such as the device model type.
Example:	`HDstring *deviceType = hdGetString(HD_DEVICE_MODEL_TYPE);`

Errors:

- HD_INVALID_INPUT_TYPE if pname does not support string as an input type.
- HD_INVALID_ENUM if pname does not support hdGet.

hdInitDevice

Description: Initializes the device. If successful, this returns a handle to the device and sets the device as the current device.

Syntax: `HHD hdInitDevice(HDstring pConfigName)`

Argument	Definition
pConfigName	The name of the device, such as the name found under the control panel “Phantom Configuration”. If HD_DEFAULT_DEVICE is passed in as pConfigName, hdInitiDevice will initialize the first device that it finds.

Returns: A handle to the initialized device.

Usage: Generally the first haptics command that should be issued.

Example: `HHD hDevice = hdInitDevice(“Default PHANToM”);`

Errors:

- HD_DEVICE_ALREADY_INITIATED if the device was already initiated.
- HD_DEVICE_FAULT if the device could not be initiated. For example, if *pConfigName* is invalid.

See also: hdDisableDevice, hdMakeCurrentDevice, hdStartScheduler

hdIsEnabled

Description: Checks if a capability is enabled. Capabilities are in hdDefines file under the enable/disable category.

Syntax: `HDboolean hdIsEnabled(HDenum cap)`

Argument	Definition
cap	Capability to check. For a list of the capabilities see Table A-9, “hdEnable, hdDisable Parameters,” on page A-10.

Returns: Whether the capability is enabled.

Usage: Capabilities are typically related to safety mechanisms. Most are turned on by default.

Example: `HDboolean bForcesOn = hdIsEnabled(HD_FORCE_OUTPUT);`

Errors:	HD_INVALID_ENUM if cap does not support enable/disable.
See also:	hdEnable, hdDisable

hdMakeCurrentDevice

Description: Makes the device current. All subsequent device-specific actions such as getting and setting state or querying device information will be performed on this device until another is made current.

Syntax:

```
void hdMakeCurrentDevice(HHD hHD)
```

Argument	Definition
hHD	The device handle of an initialized device.

Returns: void

Usage: Primarily used to switch between devices in multi-device applications.

Example:

```
hdBeginFrame(hHD1);
hdGetDoublev(HD_CURRRENT_POSITION, p1);
hdBeginFrame(hHD2);
hdGetDoublev(HD_CURRENT_POSITION, p2);
calculateforce(p1,p2,outforce);
hdMakeCurrentDevice(hHD1);
hdSetDoublev(HD_CURRENT_FORCE,outforce);
hdEndFrame(hHD1);
hdMakeCurrentDevice(hHD2);
hdSetDoublev(HD_CURRENT_FORCE,outforce);
hdEndFrame(hHD2);
```

Errors: HD_INVALID_HANDLE if the handle does not refer to an initiated device.

See also: hdInitDevice

hdSet (Parameter Values)

Description: Sets information associated with the parameter name.

Syntax:

```
void hdSetBooleanv(HDenum pname, const HDboolean *params)
void hdSetIntegerv(HDenum pname, const HDint *params)
void hdSetFloatv(HDenum pname, const HDfloat *params)
void hdSetDoublev(HDenum pname, const HDdouble *params)
void hdSetLongv (HDenum pname,const HDlong *params)
```

Argument	Definition
pname	Parameter name to set
params	Array of values to set. For a list of parameters see Table , “Set Parameters,” on page A-8.

Returns: void

Usage: Primary function for sending information to the device or changing device properties. Depending on the parameter, one should use the appropriate params type, and therefore the appropriate function signature. The caller of the function has the responsibility for allocating memory. These functions should only be called within a haptics frame.These functions should only be called within a haptics frame.

Example:

```
HDfloat forces[3] = { 2,.5, 0};
```

Or,

```
hduVector3Dd forces(0.2,0.5,0);
hdSetFloatv(HD_CURRENT_FORCE,forces);
```

Errors:

- HD_INVALID_INPUT_TYPE if pname does not support the input type.
- HD_INVALID_ENUM if pname does not support hdSet.

See also: hdGet (Parameter Values).

HD_DEVICE_ERROR

Description: A macro useful for checking if an error has occurred.

`#define HD_DEVICE_ERROR(X)(((X).errorCode)!=HD_SUCCESS)`

Syntax: `int HD_DEVICE_ERROR(x)`

Argument	Definition
X	The error structure to check. Error structure is of type HDErrorInfo

Returns: 1 if error, 0 if no error

Usage: Typically, call this with hdGetError to check if the error stack contains an error.

Example:

```
HDErrorInfo error;
if (HD_DEVICE_ERROR(error = hdGetError()))
{
   hduPrintError(stderr, &error,“HDAPI device error encountered”);
   return -1;
}
```

Errors: none

See also: hdGetError, hdGetErrorString

2

Calibration Routines

This chapter covers the routines used for managing the calibration of the device. Calibration is necessary for accurate position mapping and force/torque rendering of the haptic device. The calibration interface provides functions for querying the calibration style(s) supported by the device, checking the calibration status and updating the calibration.

The calibration styles that a particular device supports can be obtained as follows:

```
HDint supportedStyles;
hdGetIntegerv(HD_CALIBRATION_STYLE,&supportStyles);
```

It is possible for a device to support more than one calibration style. This can be masked (&) with HD_CALIBRATION_ENCODER_RESET, HD_CALIBRATION_INKWELL, HD_CALIBRATION_AUTO to determine if a particular style is supported.

hdCheckCalibration

Description: Checks the calibration status. If the return value is HD_CALIBRATION_OK if the device is calibrated. If the return value is HD_CALIBRATION_NEEDS_UPDATE, call hdUpdateCalibration to update the calibration. If the return value is HD_CALIBRATION_NEEDS_MANUAL_INPUT, the user needs to provide additional input particular to the calibration style.

Syntax: `HDenum hdCheckCalibration()`

Returns: Possible values are:

- HD_CALIBRATION_OK
- HD_CALIBRATION_NEEDS_UPDATE
- HD_CALIBRATION_NEEDS_MANUAL_INPUT

Usage: For devices that support auto calibration, call intermittently so that the device continues to update its calibration.

Example:

```
if(hdCheckCalibration()==HD_CALIBRATION_NEEDS_UPDATE)
{
    hdUpdateCalibration(myStyle);
}
```

Errors: HD_DEVICE_FAULT if the calibration information could not obtained from the device.

See Also: hdUpdateCalibration

hdUpdateCalibration

Description: Calibrates the device. The type of calibration supported can be queried through getting HD_CALIBRATION_STYLE.

Syntax:

```
void hdUpdateCalibration(HDenum style)
```

Argument	Definition
style	The calibration style of the device. For a list of calibration styles see Table A-3, “Get Identification Parameters,” on page 4.

Returns: void

Usage: Calibrate the device when hdCheckCalibration does not return HD_CALIBRATION_OK. This needs to be done once for devices that do not auto calibrate, and periodically initially for ones that do.

Example:

```
HDint style;
hdGetIntegerv(HD_CALIBRATION_STYLE,&style);
hdUpdateCalibration(style);
```

Errors: HD_DEVICE_FAULT if the calibration type could not be performed on the device

See Also: hdCheckCalibration.

3

Scheduler Routines

The following are routines for managing the scheduler and scheduler routines. This includes starting and stopping the scheduler, adding callbacks into the scheduler, and managing those callbacks.

hdGetSchedulerTimeStamp

Description: Returns the elapsed time since the start of the servo loop tick. This is useful for measuring duty cycle of the servo loop.

Syntax:

```
HDdouble hdGetSchedulerTimeStamp()
```

Returns: Number of seconds since the start of the servo loop tick.

Usage: Used to check how long operations have taken.

Example:

```
HDCallbackCode HDCALLBACK schedulerTimeCallback(void
*pUserData)
{
   HDdouble *schedulerTime = (HDdouble *)pUserData;
   *schedulerTime = hdGetSchedulerTimeStamp();
   return HD_CALLBACK_DONE;
}

HDdouble schedulerTime;
hdScheduleSynchronous(schedulerTimeCallback,&schedulerTime,
                      HD_MIN_SCHEDULER_PRIORITY);
```

Errors: none

See also: none

hdScheduleAsynchronous

Description: Schedules an operation to be executed by the scheduler in the servo loop, and does not wait for its completion. Scheduler callbacks submitted from the servo loop thread are run immediately regardless of priority.

Syntax:

```
HDSchedulerHandle
hdScheduleAsynchronous(HDSchedulerCallback pCallback,
                       void *pUserData,
                       HDushort nPriority)
```

Argument	Definition
`pCallback`	The function callback
`pUserData`	The data to be used by the function
`nPriority`	The priority of the operation. Callbacks are processed once per scheduler tick, in order of decreasing priority.

Returns: Handle for the operation.

Usage: Typically used for scheduling callbacks that run every tick of the servo loop. For example, one can run a dynamic simulation within an asynchronous callback and set forces within the simulation.

Example:

```
HDCallbackCode HDCALLBACK schedulerTimeCallback(void
*pUserData)
{
   HDint buttons;
   hdGetIntegerv(HD_CURRENT_BUTTONS,&buttons);
   if (buttons == 0)
       return HD_CALLBACK_CONTINUE;
   return HD_CALLBACK_DONE;
}

HDSchedulerHandle hHandle =
hdScheduleAsynchronous(mySchedulerCallback,
                        (void*)0,
                         HD_DEFAULT_SCHEDULER_PRIORITY);
```

Errors:

- HD_SCHEDULER_FULL if the scheduler has reached its upper limit on the number of scheduler operations that it can support at once.
- HD_INVALID_PRIORITY if nPriority is out of range,
  i.e. less than HD_MIN_SCHEDULER_PRIORITY or greater than HD_MAX_SCHEDULER_PRIORITY.

See also: HDSchedulerCallback Type, hdStartScheduler

hdScheduleSynchronous

Description: Schedules an operation to be executed by the scheduler in the servo loop, and waits for its completion. Scheduler callbacks submitted from the servo loop thread are run immediately regardless of priority.

Syntax:

```
void hdScheduleSynchronous(HDSchedulerCallback pCallback,
                           void *pUserData,
                           HDushort nPriority)
```

Argument	Definition
`pCallback`	The function callback
`pUserData`	The data to be used by the function
`nPriority`	The priority of the operation. Callbacks are processed once per scheduler tick, in order of decreasing priority.

Returns: void

Usage: Typically used as a synchronization mechanism between the servo loop thread and other threads in the application. For example, if the main application thread needs to access the position or button state, it can do so through a synchronous scheduler call. Can be used for synchronously copying state from servo loop or synchronously performing an operation in the servo loop.

Example:

```
HDCallbackCode HDCALLBACK buttonCallback(void *pUserData)
{
   int *buttons = (int *)pUserData;
   hdGetIntegerv(HD_CURRENT_BUTTONS,buttons);
   return HD_CALLBACK_DONE;
}

int buttons;
HDSchedulerHandle hHandle =
hdScheduleSynchronous(buttonCallback,
                         &buttons,
                         HD_DEFAULT_SCHEDULER_PRIORITY);
```

Errors:

- HD_SCHEDULER_FULL if the scheduler has reached its upper limit on the number of scheduler operations that it can support at once.
- HD_INVALID_PRIORITY if nPriority is out of range, i.e. less than HD_ MIN_SCHEDULER_PRIORITY or greater than HD_MAX_ SCHEDULER_PRIORITY.

See also: HDSchedulerCallback Type, hdStartScheduler

---

hdSetSchedulerRate

Description: Sets the number of times the scheduler ticks its callbacks per second.

Syntax:

```
void hdSetSchedulerRate(HDulong nRate)
```

Argument	**Definition**
nRate	The requested rate for the scheduler to run in Hz.

Returns: void

Usage: Typically used to control the fidelity of force rendering. Most haptic applications run at 1000Hz. PCI and EPP support 500, 1000, and 2000 Hz. Firewire supports 500, 1000, 1600 Hz, plus some increments in between based on the following expression: floor(8000/N + 0.5).

As a word of caution, decreasing the rate can lead to instabilities and kicking. Increasing the servo loop rate can yield stiffer surfaces and better haptic responsiveness, but leaves less time for scheduler operations to complete. When setting the scheduler rate, check hdGetSchedulerTimeStamp and HD_ INSTANTANEOUS_UPDATE_RATE to verify that the servo loop is able to maintain the rate requested.

Some devices may support a variety of settings. Certain devices may need to be flashed with the latest firmware to be able to use this feature. You can find current information from the SensAble Developer Support Center, see “Preface” for information.

Example:

```
// try to set rate of 500 Hz
hdSetScheduleRate(500);
```

Errors: HD_INVALID_VALUE if the scheduler rate specified cannot be set.

See also: hdGetSchedulerTimeStamp

hdStartScheduler

Description:	Starts the scheduler. The scheduler manages callbacks to be executed within the servo loop thread.
Syntax:	`void hdStartScheduler()`
Returns:	void
Usage:	Typically call this after all device initialization routines and after all asynchronous scheduler callbacks have been added. There is only one scheduler, so it needs to be started once, no matter how many devices one is using. Execution of callbacks starts when the scheduler is started. Forces are enabled at this point also.
Example:	`hdStartScheduler();`
Error:	HD_TIMER_ERROR if the servo loop thread could not be initialized or the servo loop could not be started.
See also:	hdStopScheduler

hdStopScheduler

Description: Stops the scheduler.

Syntax: `void hdStopScheduler()`

Returns: void

Usage: Typically call this as a first step for cleanup and shutdown of devices.

Examples:

```
hdStopScheduler();
hdUnschedule();
hdUnschedule(myCallbackHandle);
hdDisableDevice(hHD);
```

Error: HD_TIMER_ERROR if the servo loop thread could not be initialized.

See also: hdUnschedule, hdDisableDevice

hdUnschedule

Description: Un-schedules an operation by removing the associated callback from the scheduler.

Syntax:

```
void hdUnschedule(HDSchedulerHandle hHandle)
```

Argument	Definition
`hHandle`	Handle for an active operation to be unscheduled, obtained via hdScheduleAsynchronous or hdScheduleSynchronous.

Returns: void

Usage: Used to stop an active asynchronous operation. For example, if the application thread has created an asynchronous operation that returns HD_CALLBACK_CONTINUE to run indefinitely, the application can call hdUnschedule to force the callback to terminate.

Example:

```
HDSchedulerHandle hHandle =
hdScheduleAsynchronous(mySchedulerCallback,
                        (void*)0,
                       HD_DEFAULT_SCHEDULER_PRIORITY);
hdUnschedule(hHandle);
```

Errors: Generate HD_INVALID_OPERATION if the scheduler operation associated with the handle has already terminated.

See also: hdStopScheduler, hdScheduleAsynchronous, hdScheduleSynchronous

hdWaitForCompletion

Description: Checks is a callback is still scheduled for execution. This can be used as a non-blocking test or can block and wait for the callback to complete.

Syntax:

```
HDboolean hdWaitForCompletion
            (HDSchedulerHandle hHandle, HDWaitCode param)
```

Argument	Definition
`hHandle`	Handle of the target active asynchronous operation
`param`	Either HD_WAIT_CHECK_STATUS or HD_WAIT_INFINITE

Returns: Whether the callback is still scheduled.

- If HD_WAIT_CHECK_STATUS is passed into param, returns true if the scheduler operation is still active.
- If HD_WAIT_INFINITE is passed into param, this function will return true if the wait was successful or false if the wait failed.

Usage: Can be used on an asynchronous operation to either check the status or to commit to waiting for the operation to finish.

Example:

```
HDSchedulerHandle hHandle = hdSchedulerAsynchronous(
    mySchedulerCallback, (void *) 0, HD_DEFAULT_SCHEDULER_PRIORITY);

/* Do some other work and then wait for the operation to complete */

HDboolean bWaitSuccess = hdWaitForCompletion(hHandle, HD_WAIT_INFINITE);
```

Errors: none

See also: hdUnschedule, hdScheduleAsynchronous, hdScheduleSynchronous

4

HDAPI Types

The following are variable and functions types used by the HDAPI.

HDSchedulerCallback Type

Description: Used to run operations in the scheduler thread.

Syntax:

```
HDCallbackCode HDCALLBACK renderForceCallback(void
*pUserData)
{
    HDdouble beginTime = hdGetSchedulerTimeStamp();

    HHD hHD = hdGetCurrentDevice();
    hdBeginFrame(hHD);

    computeForce();

    hdEndFrame(hHD);

    HDdouble endTime = hdGetSchedulerTimeStamp();

    /* Compute the elasped time within this callback */
    HDdouble callbackTime = endTime - beginTime;
}
```

Argument	Definition
pUserData	Data used by the operation, created when the operation is scheduled.

Returns: HD_CALLBACK_DONE or HD_CALLBACK_CONTINUE depending on whether the operation is complete or should be executed again the next tick of the scheduler.

Usage: Scheduled by hdSchedulerSynchronous or hdScheduleAsynchronous, the operation is then run during the next scheduler tick, or immediately for synchronous operations if the scheduler has not been started. The return value controls whether the operation terminates after being run or runs again in the next scheduler tick. A periodic operation, i.e. one that is executed indefinitely, should return HD_CALLBACK_DONE when it is finally to be unscheduled.

HDErrorInfo Type

Description: Error structure generated by HDAPI when it reports an error.

Syntax:

```
type def struct
{
   HDerror errorCode;
   int internalErrorCode;
   HHD dHD;
} HDErrorInfo;
```

Usage: Pushed onto error stack whenever an error is generated. Query for errors by calling hdGetError. errorCode corresponds to the general error enumerates found in hdDefines.h. internalErrorCode is an error code from the underlying device drives, which useful when reporting a problem to a device vendor. hHD is the handle of the device that was active when the error occurred.

If no error has occurred, querying hdGetError will return HDError with HD_SUCCESS as the errorCode and other fields are arbitrary.

See also: hdGetError

Section II: HLAPI Routines

5

Context/Frame Management

hlBeginFrame

Description: Begins a haptics rendering pass, that is, a block of code that sends primitives to the haptic device to be rendered. hlBeginFrame updates the current state of the haptic device and clears the current set of haptics primitives being rendered in preparation for rendering a new or updated set of primitives. All haptic primitive rendering functions (that is, shapes and effects) must be made within a begin/end frame pair. hlBeginFrame also updates the world coordinate reference frame used by the haptic rendering engine. By default, hlBeginFrame samples the current GL_MODELVIEW_MATRIX from OpenGL to provide a world coordinate space for the entire haptic frame. All positions, vectors and transforms queried through hlGet* or hlCacheGet* in the client or collision threads will be transformed into that world coordinate space. Typically, the GL_MODELVIEW_MATRIX contains just the world to view transform at the beginning of a render pass.

Syntax:

```
void hlBeginFrame()
```

Returns: void

Usage: Typically, haptic primitives are specified just following (or just before) rendering of graphics primitives. hlBeginFrame is called at the start of the haptics update, the primitives are specified, and then hlEndFrame is called.

Example:

```
hlBeginFrame();
hlBeginShape(…);
…
hlEndShape();
hlEndFrame();
```

Errors: HL_INVALID_OPERATION if nested inside another hlBeginFrame or if no haptic rendering context is active.

See also: hlEndFrame

hlContextDevice

Description: Sets the haptic device for the current rendering context.

Syntax: `void hlContextDevice(HHD hHD)`

Returns: none.

Parameters: hHD **-** handle to haptic device or HD_INVALID_HANDLE to set no device.

Usage: Used to change which haptic device will be used with the current haptic rendering context.

Example:

```
HHD hHD1, hHD2;

…

// create context with first haptic device
hHLRC = hlCreateContext(hHD1);
hlMakeCurrent(hHLRC);

…

// switch to second device
hlContextDevice(hHD2);
```

Errors: HL_INVALID_OPERATION if nested inside another hlBeginFrame or if no haptic rendering context is active.

See also: hlMakeCurrent, hlCreateContext, hlDeleteContext, hlGetCurrentContext, hlGetCurrentDevice

hlCreateContext

Description: Creates a new haptic rendering context. The haptic rendering context stores the current set of haptic primitives to be rendered as well as the haptic rendering state.

Syntax: `HHLRC hlCreateContext(HHD hHD);`

Argument	Definition
hHD	Device handle to the initialized haptic device

Returns: HHLRC handle to haptic rendering context.

Usage: Called during program initialization after initializing the haptic device.

Example:

```
HDErrorInfo error;
hHD = hdInitDevice(HD_DEFAULT_DEVICE);
if (HD_DEVICE_ERROR(error = hdGetError()))
{
   hduPrintError(stderr, &error, "Failed to initialize
                 haptic device");
   exit(1);
}
hHLRC = hlCreateContext(hHD);
hlMakeCurrent(hHD, hHLRC);
```

Errors: none

See also: hlMakeCurrent, hlDeleteContext

---

hlDeleteContext

Description: Deletes a haptic rendering context. The haptic rendering context stores the current set of haptic primitives to be rendered as well as the haptic rendering state.

Syntax:

```
void hlDeleteContext(HHLRC hHLRC);
```

Argument	Definition
hHLRC	Handle to haptic rendering context returned by hlCreateContext.

Returns: none

Usage: Called at program exit to end haptic rendering and deallocate memory.

Example:

```
hHLRC = hlCreateContext(hHD);
hlMakeCurrent(hHD, hHLRC);
…
hlMakeCurrent(NULL, NULL);
hlDeleteContext(hHLRC);
```

Errors: none

See also: hlMakeCurrent, hlCreateContext

hlEndFrame

Description: Ends a haptics rendering pass, that is a block of code that sends primitives to the haptic device to be rendered. hlEndFrame flushes the set of haptics primitives (that is, shapes, effects) specified since the last hlBeginFrame to the haptics device. All haptic primitive rendering functions must be made within a begin/end frame pair.

Syntax:
```
void hlEndFrame()
```

Returns: void

Usage: Typically, haptic primitives are specified just following (or just before) rendering of graphics primitives. hlBeginFrame is called at the start of the haptics update, the primitives are specified, and then hlEndFrame is called.

Example:
```
hlBeginFrame();
hlBeginShape(...);
...
hlEndShape();
hlEndFrame();
```

Errors: HL_INVALID_OPERATION if not preceded by an hlBeginFrame, if inside a begin/end shape block, or if no rendering context is active.

See also: hlBeginFrame

hlGetCurrentContext

Description: Returns a handle to the currently active haptic rendering context.

Syntax:
```
HHLRC hlGetCurrentContext(void)
```

Returns: Handle to currently active rendering context for the current thread or NULL if no context is active.

Parameters: None.

Usage: Used to query which haptic rendering context is active for a given thread.

Example:
```
HHLRC hHLRC = hlGetCurrentContext();
if (hHLRC != myHLRC)
{
   hlMakeCurrent(myHLRC);
}
```

Errors: None.

See also: hlMakeCurrent, hlCreateContext, hlDeleteContext, hlContextDevice, hlGetCurrentContext

hlGetCurrentDevice

Description: Returns a handle to the haptic device for the currently active haptic rendering context.

Syntax: HHD hlGetCurrentDevice(void);

Returns: Handle to haptic device for currently active rendering context for the current thread or HD_INVALID_HANDLE if no context is active or no haptic device is selected for the current context.

Parameters: None.

Usage: Used to query which haptic device is active for the current context.

Example:

```
HHD hHD = hlGetCurrentDevice();
if (hHD == HD_INVALID_HANDLE)
{
   hlContextDevice(myHD);
}
```

Errors: None.

See also: hlMakeCurrent, hlCreateContext, hlDeleteContext, hlContextDevice, hlGetCurrentContext

hlMakeCurrent

Description: Makes a haptic rendering context and its associated device current. The current rendering context is the target for all rendering and state commands. All haptic rendering commands will be sent to the device in the current context until a context with a different device is made current.

Syntax:

```
void hlMakeCurrent (HHD hHD, HHLRC hHLRC)
```

Argument	**Definition**
hHD	The device handle to the initiated haptic device
hHLRC	The handle to haptic rendering context returned by hlCreateContext.

Returns: none

Usage: Called at program startup after creating the rendering context or called during program execution to switch rendering contexts in order to render to multiple haptic devices.

Example:

```
HDErrorInfo error;
hHD = hdInitDevice(HD_DEFAULT_DEVICE);
if (HD_DEVICE_ERROR(error = hdGetError()))
{
    hduPrintError(stderr, &error, "Failed to initialize
                    haptic device");
    exit(1);
}
hHLRC = hlCreateContext(hHD);
hlMakeCurrent(hHD, hHLRC);
```

Errors: none

See also: hlCreateContext, hlDeleteContext

6

State Maintenance and Accessors

hlEnable, hlDisable

Description: Enables or disables a capability for the current rendering context.

Syntax:

```
void  hlEnable(HLenum cap);
void  hlDisable(HLenum cap);
```

Argument	Definition
cap	Capability to enable. For a list of possible capabilities see, Table B-9 on page B-9.

Returns: none

Usage: Enables or disables the capability specified.

Example:

```
hlDisable(HL_PROXY_RESOLUTION);
hlProxydv(HL_PROXY_POSITION, newProxyPos);
```

Errors: HL_INVALID_ENUM if cap is not one of the values listed above.

See also: hlIsEnabled

hlGetBooleanv, hlGetDoublev, hlGetIntegerv

Description: These functions allow querying of the different state values of the haptic renderer.

Syntax:

```
void  hlGetBooleanv (HLenum pname, HLboolean *params)
void  hlGetDoublev (HLenum pname, HLdouble *params)
void  hlGetIntegerv (HLenum pname, HLint *params)
```

Argument	Definition
pname	Name of parameter (state value) to query. For a list of parameters see Table B-1, “hlGetBooleanv, hlGetDoublev,” on page B-2
params	Address at which to return the value of the parameter being queried.

Returns: none

Usage: Queries the state of the haptic renderer.

Example:

```
hlDouble proxyPos[3];
hlGetDoublev(HL_PROXY_POSITION, proxyPos);
hlBoolean switchDown;
hlGetbooleanv(HL_BUTTON1_STATE, &switchDown);
```

Errors: HL_INVALID_ENUM if pname is not one of listed values.

See also: hlIsEnabled, hlCacheGetBooleanv, hlCacheGetDoublev

hlGetError

Description: Returns the error code from the last API command called.

Syntax:

```
HLerror hlGetError(void);
```

Returns: An HLerror struct containing an HL error code and an optional device specific error code.

Usage: For a list of possible error codes the errorCode field of the returned HLerror can contain, see Table B-3, “hlGet Error,” on page B-5.

Example:

```
HLerror error = hlGetError();
if (error.errorCode == HL_DEVICE_ERROR)
{

}
```

Errors: none

hlGetString

Description: Returns strings describing the haptic renderer implementation.

Syntax:

```
const HLubyte* hlGetString (HLenum name);
```

Argument	Definition
name	Haptic renderer implementation property to describe. For a list of possible names, see Table B-4, “hlGetString,” on page B-6

Returns: A static character string describing the implementation of the haptic renderer.

Usage: Used to determine which implementation and version of the HL library that is being used. This allows for programs to take advantage of functionality in newer implementation of the library while maintaining backward compatibility.

Example:

```
const HLubyte* vendor = hlGetString(HL_VENDOR);
const HLubyte* version = hlGetString(HL_VERSION);
cout << "vender= " << vendor << "version= " << version <<
endl;
output:
vendor=SensAble Technologies, Inc. version=1.01.23
```

Errors: HL_INVALID_ENUM if name is not one of the values listed.

hlHinti, hlHintb

Description: Sets parameters that allow the haptic renderer to optionally perform selected optimizations.

Syntax:

```
void hlHinti(HLenum target, HLint value)
void hlHintb(HLenum target, HLboolean value)
```

Argument	Definition
target	Hint property to set. For list of possible Target values, see Table B-5, “hlHinti, hlHintb,” on page B-6
value	Value of target property to set

Returns: none

Usage: Used to allow control over the optimizations used by the haptics renderer.

Example:

```
hlHinti(HL_SHAPE_FEEDBACK_BUFFER_VERTICES, nVertices);
hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER);
glBegin(GL_TRIANGLES);
for (int i = 0; i < nVertices; ++i)
     glVertex3f(vertices[i][0], vertices[i][1],
                 vertices[i][2]);
glEnd();
hlEndShape();
```

Errors:

- HL_INVALID_ENUM if target is not one of the values listed above.
- HL_INVALID_VALUE if value is out of range.
- HL_INVALID_OPERATION if no haptic rendering context is current.

See also: hlBeginShape

hlIsEnabled

Description: Checks if a capability is enabled or disabled.

Syntax: `hlBoolean hlIsEnabled(HLenum cap);`

Returns: HL_TRUE if the capability is enabled in the current rendering context, HL_FALSE if it is disabled

Argument	Definition
cap	Capability to query. For a list of possible capabilities, see Table B-9, “hlEnable, hlDisable, hlIsEnabled,” on page B-9

Usage:

Example:

```
hlDisable(HL_PROXY_RESOLUTION);
assert(HL_FALSE == hlIsEnabled(HL_PROXY_RESOLUTION));
```

Errors:

- HL_INVALID_ENUM if cap is not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering is current.

7 Cached State Accessors

hlCacheGetBooleanv, hlCacheGetDoublev

Description: These functions allow querying of the different state values from a cached version of the state of the haptic renderer. Cached renderer states are passed into event and effect callback functions and contain the renderer state relevant for use by the callback function. Due to the multi threaded implementation of the haptic renderer, the state may change between the time the event occurs and the time the callback is called.

Syntax:

```
void  hlCacheGetBooleanv(HLcache* cache, HLenum pname,
                              HLboolean *params)
void  hlCacheGetDoublev(HLcache* cache HLenum pname,
                             HLdouble *params)
```

Argument	Definition
cache	state cache to query
pname	Name of parameter (state value) to query. For a list of possible pname values, see Table B-2, "hlCacheGetBooleanv, hlCacheGetDoublev," on page B-4.
params	Address at which to return the value of the parameter being queried

Returns: none

Usage: Queries the value of the state of the haptics renderer as stored in the state cache. Note that cached state is a subset of the full renderer state and therefore the possible arguments to the cached state query functions is a subset of those for the non-cached versions. The HLcache container is used by events and custom force effects. All positions, vectors and transforms accessible through hlCacheGet* are provided in world coordinates for events and workspace coordinates for custom force effects.

Example:

```
void onClickSphere(HLenum event, HLuint object,
                   HLenum thread,
                   HLcache *cache, void *userdata)
{
   hlDouble proxyPos[3];
   hlCacheGetDoublev(cache, HL_PROXY_POSITION, proxyPos);
   hlBoolean switchDown;
   hlCacheGetBooleanv(cache, HL_BUTTON1_STATE,
                      &switchDown);
}
```

Errors:

- HL_INVALID_ENUM if param is not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlGetBooleanv, hlGetDoublev, hlGetIntegerv, hlAddEventCallback

8

Shapes

hlBeginShape

Description: Indicates that subsequent geometry commands will be sent to the haptic renderer as part of the shape indicated until hlEndShape is called. Geometric primitives may only be sent to the haptic renderer if they are specified inside an hlBeginShape/hlEndShape block. Grouping geometric primitives is important for two reasons:

1 Event callbacks will report the shape id of geometric primitives touched, and

2 Correct haptic rendering of geometric primitives that are dynamically moving is accomplished by looking at frame to frame differences based on shape id.

Syntax: `void hlBeginShape(HLenum type, HLuint shape)`

Argument	Definition
type	Type of shape to specify. For a list of supported shapes types, see Table B-6, "hlBegin Shape," on page B-7
shape	The id of shape to specify returned from an earlier call hlGenShapes.

Returns: none

Usage: Used before specifying geometry and callback functions to the haptic renderer.

Example:

```
hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, myShapeId);
glBegin(GL_TRIANGLES);
glVertex3f(-1,-1,0);
glVertex3f(-1,1,0);
glVertex3f(1,1,0);
glVertex3f(1,-1,0);
glEnd();
hlEndShape();
```

Errors:

- HL_INVALID_ENUM if the type argument is not one of the values listed above.
- HL_INVALID_OPERATION if not inside an hlBeginFrame/ hlEndFrame block or if already inside an hlBeginShape/hlEndShape block.

See also: hlEndShape, hlHinti, hlHintb, hlCallback

hlDeleteShapes

Description: Deallocates unique identifiers created by hlGenShapes.

Syntax: `void hlDeleteShapes(HLuint shape, HLsizei range)`

Argument	Definition
shape	ID of first shape to delete.
range	Number of consecutive unique identifiers to generate.

Returns: none

Usage: Deletes all consecutive shape identifiers starting with shape up to shape+range.

Example:

```
HLuint myShapeId = hlGenShapes(1);
…
hlDeleteShapes(myShapeId, 1);
```

Errors:

- HL_INVALID_VALUE if any of the identifiers to deallocate were not previously allocated by glGenShapes.
- HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlGenShapes, hlBeginShape, hlIsShape

hlEndShape

Description:	Closes the shape specified by the last call to hlBeginShape. Geometry, materials, transforms and other state for the shape are captured and sent to the haptic renderer.
Syntax:	`void  hlEndShape()`
Returns:	none
Usage:	After a call to hlBeginShape.

Example:

```
hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, myShapeId);
glBegin(GL_TRIANGLES);
glVertex3f(-1,-1,0);
glVertex3f(-1,1,0);
glVertex3f(1,1,0);
glVertex3f(1,-1,0);
glEnd();
hlEndShape();
```

Errors:	HL_INVALID_OPERATION if not inside an hlBeginFrame/hlEndFrame or an hlBeginShape/hlEndShape block.
See also:	hlBeginShape

hlGenShapes

Description: For shapes, generates a unique identifier that may be used with the hlBeginShape function.

Syntax: `HLuint hlGenShapes(HLsizei range)`

Argument	Definition
range	Number of unique identifiers to generate.

Returns: A unique integer that may be used as an identifier for a shape. If the range argument is greater than one, the return value represents the first of a series of range consecutive unique identifiers.

Usage: Before a call to hlBeginShape to create a unique identifier for the new shape.

Example:

```
HLuint myShapeId = hlGenShapes(1);
hlBeginShape(HL_SHAPE_FEEDBACK_BUFFER, myShapeId);
…
hlEndShape();
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlBeginShape, hlDeleteShapes, hlIsShape

hlLocalFeature

Description: These functions specify local feature geometry to the haptic renderer.

Syntax:

```
void hlLocalFeature1fv(HLgeom *geom, HLenum type,
                       const HLfloat *v)
void hlLocalFeature1dv(HLgeom *geom, HLenum type,
                       const HLdouble *v)
void hlLocalFeature2fv(HLgeom *geom, HLenum type,
                       const HLfloat *v1,
                       const HLfloat *v2)
void hlLocalFeature2dv(HLgeom *geom, HLenum type,
                       const HLdouble *v1,
                       const HLdouble *v2)
```

Argument	Definition
geom	Container for local features passed in as parameter to callback shape closest features callback function.
ctype	Type of local feature to create. For a list of supported feature types, see Table B-8, “hlLocalFeature types,” on page B-8.
v, v1, v2	One or two vectors, given in the local coordinates of the shape, which define the local feature.

Returns: none

Usage: Called from within the closest feature callback function of a callback shape (HLclosestFeaturesProc). Used to specify the geometry of one the local features of the callback shape that is closest to the proxy position. This is used by the haptic renderer when rendering shapes using the constraint touch model. The haptic renderer will call the closest features callback in the collision thread and store the local features that are specified by that callback. It will the use these local features in the servo thread to constrain the proxy and generate forces.

Example:

```
// find the closest surface feature(s) to queryPt for sphere
// callback shape
bool closestSurfaceFeatures(const HLdouble queryPt[3],
                            const HLdouble targetPt[3],
                            HLgeom *geom,
                            void *userdata)
```

```
{
   HapticSphere *pThis = static_cast<HapticSphere *>
                          (userdata);

    // Return a plane tangent to the sphere as the closest
    // 2D local feature
   hduVector3Dd normal(queryPt);
   normal.normalize();


   hduVector3Dd point = normal * pThis->getRadius();

   hlLocalFeature2dv(geom, HL_LOCAL_FEATURE_PLANE, normal,
                     point);

   return true;
}
```

Errors: HL_INVALID_ENUM if type is not one of the values listed above.

See also: hlBeginShape, hlEndShape

---

hlIsShape

Description: Determine if an identifier is a valid shape identifier.

Syntax: `HLboolean hlIsShape(HLuint shape)`

Argument	**Definition**
shape	Identifier of shape.
range	Number of consecutive unique identifiers to generate.

Returns: Returns HL_TRUE is the shape identifier

Usage: Deletes all consecutive shape identifiers starting with shape up to shape+range.

Example:

```
HLuint myShapeId = hlGenShapes(1);
assert(hlIsEffect(myShapeId));
```

Errors: HL_INVALID_OPERATION if called within an hlBeginShape/ hlEndShape block.

See also: hlGenShapes, hlBeginShape, hlIsShape

hlShapeGetBooleanv, hlShapeGetDoublev

Description: These functions allow querying of the state of specific shapes.

Syntax:

```
void hlGetShapeBooleanv(HLuint shapeId, HLenum pname,
                           HLboolean *params)
void hlGetShapeDoublev(HLuint shapeId, HLenum pname,
                          HLdouble *params)
```

Argument	**Definition**
shapeid	Identifier of shape to query
pname	name of parameter (state value) to query
params	address at which to return the value of the parameter being queried. For a list of supported shapes types, see Table B-7, “hlShapeGetBooleanv, hlShapeGetDoublev,” on page B-7.

Returns: none

Usage: Queries the state of the shape with identifier shapeid from the haptic rendering engine. This shape must have been rendered during the last frame.

Example:

```
HLboolean isTouching;
hlGetShapeBooleanv(myShapeId, HL_PROXY_IS_TOUCHING,
                   &isTouching);
if (isTouching)
{
    hduVector3Dd force;
    hlGetShapeDoublev(myShapeId, HL_REACTION_FORCE, force);
}
```

Errors:

- HL_INVALID_ENUM if param is not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlBeginShape, hlEndShape, hlGetBooleanv, hlGetDoublev, hlGetIntegerv

9

Material and Surface Properties.

hlGetMaterialfv

Description: Gets the current haptic material properties for shapes and constraints.

Syntax:

```
void hlGetMaterialfv(HLenum face, HLenum pname, HLfloat
*params)
```

Argument	Definition
face	Face(s) to apply this material to. For a list of the supported face values, see Table B-12, “hlGetMaterialfv - face values,” on page B-10.
pname	Material property to set. For a list of the supported values for pname, see Table B-13, “hlGetMaterialfv - pname values,” on page B-10.
param	New value for material property.

Returns: none

Usage: Used to find the material properties that will be used for rendering shapes and constraints.

Example:

```
hlFloat stiffness;
hlGetMaterialfv(HL_FRONT, HL_STIFFNESS, &stifness);
```

Errors:

- HL_INVALID_ENUM if the face or pname arguments are not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is current.

See also:	hlMaterialf

---

hlMaterialf

Description:	Sets haptic material properties for shapes and constraints. Material properties can be set independently for front and back facing triangles of contact shapes. Only front face materials apply to constraints.

Syntax:

```
void hlMaterialf(HLenum face, HLenum pname, HLfloat param)
```

Argument	Definition
face	Face(s) to apply this material to. For a list of the supported face values, see Table B-10, “hlMaterialf - face values,” on page B-9.
pname	Material property to set. For a list of the supported values for pname, see Table B-11, “hlMaterialf - pname values,” on page B-10.
param	New value for material property.

Returns:	none
Usage:	Used before defining shapes to set their materials.

Example:

```
hlMaterial(HL_FRONT_AND_BACK, HL_STIFFNESS, 0.8);
hlMaterial(HL_FRONT_AND_BACK, HL_DAMPING, 0.6);
hlMaterial(HL_FRONT_AND_BACK, HL_STATIC_FRICTION, 0.5);
hlMaterial(HL_FRONT_AND_BACK, HL_DYNAMIC_FRICTION, 0.4);
```

Errors:

- HL_INVALID_ENUM if the face or pname arguments are not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is current.
- HL_INVALID_VALUE if param is not between 0 and 1.

See also:	hlGetMaterialfv

hlTouchableFace

Description: Sets which faces of shapes will be touchable by the haptic device.

Syntax: `void hlTouchableFace(HLenum mode)`

Argument	**Definition**
mode	Which face(s) of shapes to make touchable. For a list of mode values, see Table B-14, "hlTouchableFace - mode values," on page B-11

Returns: none

Usage: Shapes may be touchable on one or both sides. Use this function to set which of those sides is touchable. The front side of feedback buffer and depth buffer shapes is defined by the winding order of the vertices as set in OpenGL; that is, triangles considered front facing by OpenGL will also be considered as front facing by HL.

When using the HL_CONSTRAINT touch model, all shapes are always touchable from both sides, independent of the touchable face.

Example: `hlTouchableFace(HL_FRONT);`

Errors:

- HL_INVALID_ENUM if the mode argument is not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlTouchModel

hlTouchModel

Description: Sets the touch model to specify shapes as contact shapes or constraints.

Syntax: `void hlTouchModel(HLenum mode)`

Argument	Definition
mode	Contact or constraint model. For a list of supported mode values, see Table B-15, “hlTouchModel,” on page B-11.

Returns: none

Usage: Used before specifying a shape to set whether it is a standard contact shape or whether it is a constrain that will force the haptic device to the surface of the shape.

Example: `hlTouchModel(HL_CONTACT);`

Errors:
- HL_INVALID_ENUM if mode is not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is current.

See also: hlTouchModelf

hlTouchModelf

Description: Sets properties of the touch model.

Syntax: `void hlTouchModelf(HLenum pname, HLfloat param)`

Argument	Definition
pname	Touch model parameter to modify. For a list of supported values for pname, see Table B-16, “hlTouchModelIf,” on page B-11.
param	New value for the parameter

Returns: none

Usage: Used before specifying a shape to set the parameters used by the touch model for that shape.

Example: `hlTouchModelf(HL_SNAP_DISTANCE, 1.5);`

Errors:

- HL_INVALID_ENUM if pname is not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is current.

See also: hlTouchModel

10

Force Effects

hlDeleteEffects

Description: For effect, deallocates unique identifiers created by hlGenEffects.

Syntax: `void hlDeleteEffects(HLuint` *`effect`*`, HLsizei` *`range`*`)`

Argument	Definition
effect	Identifier of first effect to delete
range	Number of consecutive unique identifiers to generate

Returns: none

Usage: Deletes all consecutive effect identifiers starting with effect up to effect+range.

Example:

```
HLuint myEffectId = hlGenEffects(1);
…
hlDeleteEffects(myEffectId, 1);
```

Errors:

- HL_INVALID_VALUE if any of the identifiers to deallocate were not previously allocated by glGenEffects.
- HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlGenEffects, hlStartEffect, hlTriggerEffect, hlIsEffect

hlEffectd, hlEffecti, hlEffectdv, hlEffectiv

Description: Sets the current value of an effect property.

Syntax:

```
void hlEffectd (HLenum pname, HLdouble param)
void hlEffecti (HLenum pname, HLint param)
void hlEffectdv (HLenum pname, const HLdouble *params)
void hlEffectiv (HLenum pname, const HLint *params)
```

Argument	**Definition**
pname	The name of parameter to set. For a list of possible names, see Table B-19, “hlEffectd, hlEffecti, hlEffectdv, hlEffectiv,” on page B-13
params	The new value of the property.

Returns: none

Usage: Sets the value of an effect property which will be applied to the effect generated by the next call to hlStartEffect or hlTriggerEffect.

Example:

```
hlBeginFrame();
    hlEffectd(HL_EFFECT_PROPERTY_DURATION, pulseLength);
    hlEffectd(HL_EFFECT_PROPERTY_GAIN, 0.4);
    hlEffectd(HL_EFFECT_PROPERTY_MAGNITUDE, 0.4);
    hlTriggerEffect(HL_EFFECT_FRICTION);
hlEndFrame();
```

Errors:

- HL_INVALID_ENUM if the type argument is not one of the values listed above.
- HL_INVALID_OPERATION if not inside an hlBeginFrame/ hlEndFrame block or if inside an hlBeginShape/hlEndShape block.

See also: hlStartEffect, hlTriggerEffect, hlStartEffect, hlEffectd, hlEffecti, hlEffectdv, hlEffectiv, hlCallback,

hlGenEffects

Description: Generates unique identifiers for effects that may be used with the hlStartEffect function.

Syntax:

```
HLuint hlGenEffects(HLsizei range)
```

Argument	Definition
range	Number of unique identifiers to generate

Returns: A unique integer that may be used as an identifier for an effect. If the range argument is greater than one, the return value represents the first of a series of range consecutive unique identifiers.

Usage: Before a call to hlStartEffect to create a unique identifier for the new effect.

Example:

```
friction = hlGenEffects(1);
hlBeginFrame();
hlEffectd(HL_EFFECT_PROPERTY_GAIN, 0.4);
hlEffectd(HL_EFFECT_PROPERTY_MAGNITUDE, 0.4);
hlStartEffect(HL_EFFECT_FRICTION, friction);
hlEndFrame();
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active or if inside an hlBeginShape/hlEndShape block.

See also: hlStartEffect, hlStopEffect, hlDeleteEffects, hlIsEffect

hlGetEffectdv, hlGetEffectiv

Description: Queries the current value of an effect property.

Syntax:

```
void hlGetEffectdv (HLuint effect, HLenum pname,
                    HLdouble *params)
void hlGetEffectiv (HLuint effect, HLenum pname,
                    HLint *params)
```

Argument	Definition
effect	Identifier of effect to query.
pname	Name of parameter to query. For a list of supported name values, see Table B-20, "hlGetEffectdv, hlGetEffectiv," on page B-14.
params	Receives the value of the property.

Returns: none

Usage: Gets the value of an effect property which will be applied to the effect generated by the next call to hlStartEffect or hlTriggerEffect.

Example:

```
hlBeginFrame();
    hlEffectd(HL_EFFECT_PROPERTY_GAIN, 0.4);
    hlStartEffect(myEffect, HL_EFFECT_FRICTION);
hlEndFrame();

HLdouble gain;
hlGetEffectdv(myEffect, HL_EFFECT_PROPERTY_GAIN, gain);
assert(gain == 0.4);
```

Errors:

- HL_INVALID_ENUM if the name argument is not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlStartEffect, hlTriggerEffect, hlEffectd, hlEffecti, hlEffectdv, hlEffectiv

hlIsEffect

Description: Determine if an identifier is a valid effect identifier.

Syntax: `HLboolean hlIsEffect(HLuint effect)`

Argument	Definition
effect	Identifier of first effect to check

Returns: none

Usage: Returns true if effect is a valid identifier that was previously returned by hlGenEffects.

Example:

```
HLuint myEffectId = hlGenEffects(1);
…
assert(hlIsShape(myEffectId);
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlGenEffects

hlStartEffect

Description: Starts an effect which will continue to run until it is terminated by a call to hlStopEffect.

Syntax: `void hlStartEffect (HLenum type, HLuint effect)`

Argument	Definition
type	Type of effect to start. For a list of supported types, see Table B-17, “hlStartEffect - effect types,” on page B-12.
effect	Identifier of effect to start, generated by a call to hlGenEffects.

Returns: none

Usage: Starting an effect will cause it to continue to run until hlStopEffect is called to terminate it. When using hlStartEffect, the duration property is ignored.

Example:

```
hlBeginFrame();
    hlEffectd(HL_EFFECT_PROPERTY_DURATION, pulseLength);
    hlEffectd(HL_EFFECT_PROPERTY_GAIN, 0.4);
    hlEffectd(HL_EFFECT_PROPERTY_MAGNITUDE, 0.4);
    hlStartEffect(HL_EFFECT_FRICTION);
hlEndFrame();
```

Errors:

- HL_INVALID_ENUM if the type argument is not one of the values listed above.
- HL_INVALID_OPERATION if not inside an hlBeginFrame/ hlEndFrame block or if inside an hlBeginShape/hlEndShape block.

See also: hlStopEffect, hlEffectd, hlEffecti, hlEffectdv, hlEffectiv, hlTriggerEffect, hlCallback

hlStopEffect

Description: Stops an effect that was started with hlStartEffect.

Syntax:

```
void hlStopEffect (HLuint effect);
```

Argument	Definition
effect	Identifier of effect to stop, generated by a call to hlGenEffects.

Returns: none

Usage: Once an effect has been started by a call hlStartEffect, it will continue running until hlStopEffect is called.

Example:

```
hlBeginFrame();
hlStopEffect(friction);
hlEndFrame();
```

Errors: HL_INVALID_OPERATION if not inside an hlBeginFrame/hlEndFrame block or if inside an hlBeginShape/hlEndShape block.

See also: hlStartEffect, hlEffectd, hlEffectdv, hlEffecti, hlEffectiv, hlTriggerEffect, hlCallback

hlTriggerEffect

Description: Starts an effect which will continue to run for a specified duration.

Syntax:

```
void hlTriggerEffect (HLenum type);
```

Argument	Definition
type	Type of effect to start. For a list of support value types, see Table B-18, “hlTriggerEffect,” on page B-13.

Returns: none

Usage: Triggering an effect will cause it to continue to run for the amount of time specified by the current effect duration. Unlike effects started with hlStartEffect, those started with hlTriggerEffect do not need to be terminated with a call to hlStopEffect. The effect will be terminated automatically when the time elapsed since the call to hlTriggerEffect becomes greater than the effect duration. The effect duration will be the value specified by the last call to hlEffectd with HL_EFFECT_PROPERTY_DURATION.

Example:

```
hlBeginFrame();
    hlEffectd(HL_EFFECT_PROPERTY_DURATION, pulseLength);
    hlEffectd(HL_EFFECT_PROPERTY_GAIN, 0.4);
    hlEffectd(HL_EFFECT_PROPERTY_MAGNITUDE, 0.4);
    hlTriggerEffect(HL_EFFECT_FRICTION);
hlEndFrame();
```

Errors:

- HL_INVALID_ENUM if the type argument is not one of the values listed above.
- HL_INVALID_OPERATION if not inside an hlBeginFrame/ hlEndFrame block or if inside an hlBeginShape/hlEndShape block.

See also: hlStartEffect, hlEffectd, hlEffecti, hlEffectdv, hlEffectiv, hlCallback

11

Proxy

hlProxydv

Description: Sets properties of the proxy.

Syntax: `void hlProxydv (HLenum pname, const HLdouble *params)`

Argument	Definition
pname	Name of parameter to set. For a list of supported pnames, see Table B-25, “hlProxydv,” on page B-17.
param	Value of the property to set.

Returns: none

Usage: Sets properties of the proxy. The proxy is a virtual object that follows the haptic device but is constrained by geometric primitives. The force sent to the haptic device is based in part on the relative positions of the haptic device and proxy. When proxy resolution is enabled, the haptic rendering engine updates the proxy position automatically as the haptic device moves and geometric primitives change. Users may disable automatic proxy resolution (see hlDisable) and set the proxy position directly in order to use their own algorithms for proxy resolution.

Example:

```
hlBeginFrame();
hlDisable(HL_PROXY_RESOLUTION);
HLdouble newProxyPos[] = {3, 4, 5};
hlProxydv(HL_PROXY_POSITION, newProxyPos);
hlEndFrame();
```

Errors:

- HL_INVALID_ENUM if the name argument is not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is active or if not within an hlBeginFrame/hlEndFrame block.

See also: hlEnable, hlDisable, hlGetDoublevhlGetDoublev

12 Transforms

hlLoadIdentity

Description: Replaces the current matrix on the top of the current matrix stack with the identity matrix.

Syntax: `void hlLoadIdentity(void)`

Returns: none

Usage: Clears the top of the current matrix and replaces it with the identity matrix:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This command applies to either the touchworkspace or viewtouch matrix depending on the current matrix mode.

Example:

```
hlMatrixMode(HL_TOUCHWORKSPACE);
hlPushMatrix(); // save off old touchworkspace matrix
hlLoadIdentity(); // set touchworkspace to identity
hlScaled(wsscale, wsscale, wsscale); // set scale
// render some stuff
…
hlPopMatrix(); // restore old touchworkspace matrix
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlMatrixMode, hlPushMatrix, hlPopMatrix, hlTranslated, hlTranslatef, hlRotated, hlRotatef, hlScaled, hlScalef, hlLoadMatrixd, hlLoadMatrixf, hlMultMatrixd, hlMultMatrixf, hlGetBooleanv, hlGetDoublev, hlGetIntegerv

hlLoadMatrixd, hlLoadMatrixf

Description: Replaces the current matrix on the top of the current matrix stack with the 4x4 matrix specified.

Syntax:

```
void hlLoadMatrixd(const HLdouble *m)
void hlLoadMatrixf(const HLfloat *m)
```

Argument	Definition
m	An array of 16 floating point or double precision values representing a 4x4 transform matrix.

Returns: none

Usage: Clears the top of the current matrix and replaces it with the 4x4 matrix constructed from the values in m as follows:

$$\begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

This command applies to either the touchworkspace or viewtouch matrix, depending on the current matrix mode.

Example:

```
HLdouble myShapeXfm[16];
…
hlPushMatrix(); // save off old matrix
hlLoadMatrixd(myShapeXfm); // set xfm
// render some stuff
…
hlPopMatrix(); // restore old matrix
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlMatrixMode, hlPushMatrix, hlPopMatrix, hlTranslated, hlTranslatef, hlRotated, hlRotatef, hlScaled, hlScalef, hlLoadIdentity, hlMultMatrixd, hlMultMatrixf, hlGetBooleanv, hlGetDoublev, hlGetIntegerv

hlMultMatrixd, hlMultMatrixf

Description: Multiplies the current matrix on the top of the current matrix stack with the 4x4 matrix specified.

Syntax:

```
void hlMultMatrixd(const HLdouble *m)
void hlMultMatrixf(const HLfloat *m)
```

Argument	Definition
m	An array of 16 floating point or double precision values representing a 4x4 transform matrix.

Returns: none

Usage: Replaces the top of the current matrix stack with the product of the top of the stack and the matrix constructed from the values in m as follows:

$$\begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

This command applies to either the touchworkspace or viewtouch matrix, depending on the current matrix mode.

Example:

```
HLdouble myShapeXfm[16];
…
hlPushMatrix(); // save off old matrix
hlMultMatrixd(myShapeXfm); // set xfm
// render some stuff
…
hlPopMatrix(); // restore old matrix
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlMatrixMode, hlPushMatrix, hlPopMatrix, hlTranslated, hlTranslatef, hlRotated, hlRotatef, hlScaled, hlScalef, hlLoadIdentity, hlLoadMatrixd, hlLoadMatrixf, hlGetBooleanv, hlGetDoublev, hlGetIntegerv

hlMatrixMode

Description: Sets which matrix stack is the target for future calls to matrix manipulation commands.

Syntax:

```
void hlMatrixMode(HLenum mode);
```

Argument	Definition
mode	Which matrix stack to apply matrix manipulation commands to. See Table B-26, “hlMatrix - mode values,” on page B-17 in Appendix B for a list of valid modes.

Returns: none

Usage: The HLAPI maintains two transforms that, when multiplied together, determine the mapping from the local coordinate system of the haptic device (workspace coordinates) to graphical view coordinates. The first transform maps from touch coordinates to workspace coordinates and the second maps from view coordinates to touch coordinates. The API maintains a stack of matrices for each of these of two transforms where the top of the stack represents the current matrix to use for the transform. All matrix manipulation commands target the top matrix on one of these two stacks. Setting the matrix mode controls which of the two stacks is targeted.

Example:

```
hlMatrixMode(HL_TOUCHWORKSPACE);
hlPushMatrix(); // save off old touchworkspace matrix
hlLoadIdentity(); // set touchworkspace to identity
hlScaled(wsscale, wsscale, wsscale); // set scale
// render some stuff
…
hlPopMatrix(); // restore old touchworkspace matrix
```

Errors:

- HL_INVALID_ENUM if the mode argument is not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlPushMatrix, hlPopMatrix, hlTranslatef, hlTranslated, hlRotatef, hlRotated, hlScalef, hlScaled, hlWorkspace, hlLoadMatrixd, hlLoadMatrixf hlMultMatrixd, hlMultMatrixf, hlGetBooleanv, hlGetDoublev, hlGetIntegerv

hlOrtho

Description: Sets up the haptic view volume which determines how the workspace of the haptic device will be mapped into the graphical view volume.

Syntax:

```
void hlOrtho (HLdouble left, HLdouble right,
               HLdouble bottom, HLdouble top,
               HLdouble zNear, HLdouble zFar);
```

Argument	Definition
left, right	The coordinates of the left and right boundaries of the haptic view volume
top, bottom	the coordinates of the top and bottom boundaries of the haptic view volume.
zNear, zFar	the coordinates of the front and back boundaries of the haptic view volume.

Returns: none

Usage: The hlOrtho function is used in conjunction with hlWorkspace to determine the mapping between the physical workspace of the haptic device and the graphical view. hlOrtho defines an orthogonal view volume (that is an axis oriented bounding box) in the graphical view coordinate system. hlWorkspace defines a corresponding box in the physical coordinates of the haptic device. The haptic rendering engine uses these to determine a transformation between the two coordinate spaces. Specifically, the API creates a transform consisting of a uniform or non-uniform scale about the center of the workspace box and a translation. The product of this transform and the top of the matrix stack replaces the current top of the matrix stack. This function is generally used with the HL_TOUCHWORKSPACE matrix mode.

Example:

```
hlMatrixMode(HL_TOUCHWORKSPACE);

// clear the matrix stack
hlLoadIdentity();

// specify the boundaries for the workspace of the haptic
// device in millimeters in the cordiantes of the haptic
// device the haptics engine will map the view volume to
// this workspace
```

```
hlWorkspace (-80, -80, -70, 80, 80, 20);
// specify the haptic view volume
hlOrtho (0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlWorkspace, hlMatrixMode

hlPushMatrix, hlPopMatrix

Description: hlPushMatrix pushes a new matrix onto the top of the current matrix stack. hlPopMatrix removes the top of the current matrix stack.

Syntax:

```
void hlPushMatrix()
void hlPopMatrix()
```

Returns: none

Usage: The HLAPI maintains matrix stacks, one for the viewtouch matrix and the other for the touchworkspace matrix. hlPushMatrix and hlPopMatrix allow these matrix stacks to be changed temporarily and then restored. hlPushMatrix pushes a new matrix onto the top of the current matrix stack. The new matrix is initially a duplicate of the existing matrix on the top of the stack. hlPopMatrix removes the matrix on the top of the stack, leaving the old top of the stack as the current matrix.

This command applies to either the touchworkspace or viewtouch matrix depending on the current matrix mode.

Example:

```
hlPushMatrix(); // save off old matrix
hlTranslate(0, 20, 0); // move geometry up 20
// draw some stuff
hlPopMatrix(); // restore old matrix
```

Errors:

- HL_INVALID_OPERATION if no haptic rendering context is active.
- HL_STACK_OVERFLOW if hlPushMatrix is called when the matrix stack is already full.
- HL_STACK_UNDERFLOW if hlPopMatrix is called when the matrix stack is only one deep.

See also: hlMatrixMode, hlTranslated, hlTranslatef, hlRotated, hlRotatef, hlScaled, hlScalef, hlLoadIdentity, hlLoadMatrixd, hlLoadMatrixf, hlMultMatrixd, hlMultMatrixf, hlGetBooleanv, hlGetDoublev, hlGetIntegerv

hlRotatef, hlRotated

Description: Multiplies the current matrix on the top of the current matrix stack by a 4x4 rotation matrix.

Syntax:

```
void hlRotated (HLdouble angle, HLdouble x, HLdouble y,
                    HLdouble z)
void hlRotatef (HLfloat angle, HLfloat x,
                    HLfloat y,HLfloatz)
```

Argument	Definition
angle	angle, in degrees, to rotate.
x, y, z	coordinates of a vector representing the axis to rotate about.

Returns: none

Usage: Creates a 4x4 rotation matrix that represents a rotation of angle degrees about the given axis. Replaces the top of the current matrix stack with the product of the top of the matrix stack and the rotation matrix. The rotation matrix created is of the form:

$$\begin{pmatrix} x^2+\cos(angle)(1-x^2) & xy[1-\cos(angle)]-z\sin(angle) & zx[1-\cos(angle)]+y\sin(angle) & 0 \\ xy[1-\cos(angle)]+z\sin(angle) & y^2+\cos(angle)(1-y^2) & yz[1-\cos(angle)]-x\sin(angle) & 0 \\ zx[1-\cos(angle)]-y\sin(angle) & yz[1-\cos(angle)]+x\sin(angle) & z^2+\cos(angle)(1-z^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This command applies to either the touchworkspace or viewtouch matrix depending on the current matrix mode.

Example:

```
hlPushMatrix(); // save off old matrix
hlRotate(45, 0, 0, 1); // rotate 45 degrees about z axis
// draw some stuff
hlPopMatrix(); // restore old matrix
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlMatrixMode, hlPushMatrix, hlPopMatrix, hlTranslated, hlTranslatef, hlScaled, hlScalef, hlLoadIdentity, hlLoadMatrixd, hlLoadMatrixf, hlMultMatrixd, hlMultMatrixf, hlGetBooleanv, hlGetDoublev, hlGetIntegerv

hlScalef, hlScaled

Description: Multiplies the current matrix on the top of the current matrix stack by a 4x4 scale matrix.

Syntax:

```
void hlScaled(HLdouble x, HLdouble y, HLdouble z);
void hlScalef(HLfloat x, HLfloat y, HLfloat z);
```

Argument	Definition
x, y, z	scale factors about the x, y and z axes respectively.

Returns: none

Usage: Creates a 4x4 scale matrix that scales about the three coordinate axes. Replaces the top of the current matrix stack with the product of the top of the matrix stack and the scale matrix. The scale matrix created is of the form:

$$\begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This command applies to either the touchworkspace or viewtouch matrix depending on the current matrix mode.

Example:

```
hlPushMatrix(); // save off old matrix
hlScale(1, 2, 1); // scale 2x along y axis
// draw some stuff
hlPopMatrix(); // restore old matrix
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlMatrixMode, hlPushMatrix, hlPopMatrix, hlTranslated, hlTranslatef, hlRotated, hlRotatef, hlLoadIdentity, hlLoadMatrixd, hlLoadMatrixf, hlMultMatrixd, hlMultMatrixf, hlGetBooleanv, hlGetDoublev, hlGetIntegerv

hlTranslatef, hlTranslated

Description: Multiplies the current matrix on the top of the current matrix stack by a 4x4 translation matrix.

Syntax:

```
void hlTranslated (HLdouble x, HLdouble y, HLdouble z)
void hlTranslatef (HLfloat x, HLfloat y, HLfloat z)
```

Argument	Definition
x, y, z	Coordinates of translation vector.

Returns: none

Usage: Creates a 4x4 translation matrix based on the vector (x, y, z) and replaces the top of the current matrix stack with the product of the top of the matrix stack and the translation matrix. The translation matrix created is of the form:

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This command applies to either the touchworkspace or viewtouch matrix depending on the current matrix mode.

Example:

```
hlPushMatrix(); // save off old matrix
hlTranslate(0, 20, 0); // move geometry up 20
// draw some stuff
hlPopMatrix(); // restore old matrix
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlMatrixMode, hlPushMatrix, hlPopMatrix, hlRotated, hlRotatef, hlScaled, hlScalef, hlLoadIdentity, hlLoadMatrixd, hlLoadMatrixf, hlMultMatrixd, hlMultMatrixf, hlGetBooleanv, hlGetDoublev, hlGetIntegerv

hlWorkspace

Description: Defines the extents of the workspace of the haptic device that will be used for mapping between the graphics coordinate system and the physical units of the haptic device.

Syntax:

```
void hlWorkspace (HLdouble left, HLdouble bottom, HLdouble
back, HLdouble right, HLdouble top, HLdouble front);
```

Argument	Definition
left, right	The coordinates of the left and right boundaries of the haptic device workspace.
top, bottom	the coordinates of the top and bottom boundaries of the haptic view volume.
front, back	the coordinates of the front and back boundaries of the haptic view volume.

Returns: none

Usage: The hlWorkspace function is used in conjunction with hlOrtho to determine the mapping between the physical workspace of the haptic device and the graphical view. hlOrtho defines an orthogonal view volume (that is an axis oriented bounding box) in the graphical view coordinate system. hlWorkspace defines a corresponding box in the physical coordinates of the haptic device. The haptic rendering engine uses these to determine a transformation between the two coordinate spaces. Specifically, the API creates a transform consisting of a uniform or non-uniform scale about the center of the workspace box and a translation. The product of this transform and the top of the matrix stack replaces the current top of the matrix stack. This function is generally used with the HL_TOUCHWORKSPACE matrix mode.

Example:

```
hlMatrixMode(HL_TOUCHWORKSPACE);

// clear the matrix stack
hlLoadIdentity();

// specify the boundaries for the workspace of the haptic
// device in millimeters in the coordinates of the haptic
// device the haptics engine will map the view volume to
// this workspace
hlWorkspace (-80, -80, -70, 80, 80, 20);
```

```
// specify the haptic view volume
hlOrtho (0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is active.

See also: hlOrtho, hlMatrixMode

13

Callbacks

hlCallback

Description: Sets a user callback function.

Syntax:

```
void hlCallback (HLenum type, HLcallbackProc fn, void
*userdata)
```

Argument	Definition
type	Name of callback to set. For a list of supported type values, see Table B-21, “hlCallback,” on page B-15.
fn	Pointer to callback function.
userdata	Pointer to client specific data that will be passed to the callback function.

Returns: none

Usage: Used to set callback functions for custom shapes and custom effect types.

Example:

```
hlBeginShape(HL_SHAPE_CALLBACK, myshape);
hlCallback(HL_SHAPE_INTERSECT_LS, HLcallbackProc)
           &intersectSurface, (void*) &myShapeData);
hlCallback(HL_SHAPE_CLOSEST_POINT, (HLcallbackProc)
            &closestPointSurface, (void*) &myShapeData);
hlEndShape();
```

Errors:

- HL_INVALID_ENUM if type is not one of the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is current.

See also: hlBeginShape

14

Events

hlAddEventCallback

Description: Adds a user defined event handling function to the list of callback functions for an event.

Syntax:

```
void hlAddEventCallback (HLenum event, HLuint shapeId,
                         HLenum thread, HLeventProc fn,
                         void *userdata)
```

Argument	Definition
event	Event to subscribe to. For a list of supported event value, see Table B-22, “hlAddEventCallback, hlRemoveEventCallBack - event values:,” on page B-16.
shapeId	Identifier of shape. Callback will only be called if the event occurs on the shape with this identifier unless this argument is set HL_OBJECT_ANY in which case the callback will be called independent of any objects.
thread	Thread to have callback function called in. For a list of support thread values, see Table B-23, “hlAddEventCallback, hlRemoveEventCallback - thread values,” on page B-16.
fn	Pointer to callback function
userdata	Pointer to client specific data that will be passed to the callback function.

Returns: none

Usage: Event callbacks are used to inform programs about occurrences in the haptic renderer such as an object being touched

Example:

```
void HLCALLBACK onClickSphere(HLenum event, HLuint object,
                              HLenum thread,
                              HLcache *cache,
                              void *userdata)
{
   // handle event
}

AddEventCallback(HL_EVENT_1BUTTONDOWN, mySphereId,
                 HL_CLIENT_THREAD, &onClickSphere,
                 &mydata);
```

Errors:

- HL_INVALID_ENUM if event and thread are not among the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is current.

See also: hlRemoveEventCallback, hlEventd, hlCheckEvents

hlCheckEvents

Description: Calls callback functions for all events that are subscribed to and that have occurred since the last call to hlCheckEvents.

Syntax: `void hlCheckEvents()`

Returns: none

Usage: Should be called on a regular basis in the client thread. This is often called as part of the main rendering loop or main event loop. This function checks if any subscribed events have been triggered since the last call to hlCheckEvents. If any events have been triggered, then event callbacks for these events will be called synchronously before the call to hlCheckEvents returns. This only applies to event callbacks in the client thread (HL_THREAD_CLIENT), collision thread events will be called from within the collision thread, outside of calls to hlCheckEvents.

Example:

```
void HLCALLBACK onMotion(HLenum event, HLuint object,
                         HLenum thread, HLcache *cache,
                         void *userdata)
{
   hlBeginFrame();
   drawScene();
   hlEndFrame();
}

void setup()
{
   hlAddEventCallback(HL_EVENT_MOTION, HL_OBJECT_ANY,
                      HL_CLIENT_THREAD, &onMotion, NULL);
}

void onIdle()
{
   hlCheckEvents();
}
```

Errors: HL_INVALID_OPERATION if no haptic rendering context is current.

See also: hlAddEventCallback, hlRemoveEventCallback, hlEventd

hlEventd

Description: Sets parameters that influence how and when event callbacks are called.

Syntax: `void hlEventd(HLenum pname, HLdouble param)`

Argument	Definition
pname	Name of event parameter to set. For a list of pname parameters, see Table B-24, “hlEventd - pname values,” on page B-17.
param	Value of parameter to set.

Returns: none

Usage: Used at program startup to tailor the event management of the haptic renderer to the specific application.

Example: `hlEventd(HL_EVENT_MOTION_LINEAR_TOLERANCE, 10);`

Errors:

- HL_INVALID_ENUM if pname is not one of the values listed above.
- HL_INVALID_VALUE if value is out of range.
- HL_INVALID_OPERATION if no haptic rendering context is current.

See also: hlAddEventCallback

hlRemoveEventCallback

Description: Removes an existing user defined event handling function from the list of callback functions for an event.

Syntax:

```
void hlRemoveEventCallback (HLenum event,
                            HLuint shapeId,
                            HLenum thread,
                            HLeventProc fn)
```

Argument	Definition
event	Subscribed event. For a list of event parameters, see Table B-22, “hlAddEventCallback, hlRemoveEventCallBack - event values:,” on page B-16.
shapeId	Identifier of shape. Callback will only be called if the event occurs on the shape with this identifier unless this argument is set: HL_OBJECT_ANY. In which case the callback will be called regardless of what object is being touched.
thread	Thread in which to have callback function called. For a list of thread parameters, see Table B-23, “hlAddEventCallback, hlRemoveEventCallback - thread values,” on page B-16.

Returns: none

Usage: Removes any event callback that was added to list of subscribed events with a call to hlAddEventCallback with the same event, thread, shapeId and function pointer.

Example:

```
void HLCALLBACK onClickSphere(HLenum event, HLuint object,
                              HLenum thread,
                              HLcache *cache,
                              void *userdata)
{
   // handle event
}

AddEventCallback(HL_EVENT_1BUTTONDOWN, mySphereId,
                 HL_CLIENT_THREAD, &onClickSphere,
                 &mydata);
RemoveEventCallback(HL_EVENT_1BUTTONDOWN, mySphereId,
                    HL_CLIENT_THREAD, &onClickSphere);
```

Errors:

- HL_INVALID_ENUM if event and thread are not among the values listed above.
- HL_INVALID_OPERATION if no haptic rendering context is current.

See also: hlAddEventCallback, hlCheckEvents, hlEventd

15

Calibration

hlUpdateCalibration

Description: Causes the haptic device to re calibrate.

Syntax: `void hlUpdateCalibration(void)`

Returns: none

Usage: Calibration of the haptic device is a two-stage process. First, new calibration data must be found. How this is done depends on the device hardware. For the PHANTOM Omni device, new calibration data is found when the stylus is inserted into the inkwell. For the PHANTOM Desktop device, calibration data is found as the device is moved about the extents of the workspace. A program may subscribe to the HL_EVENT_CALIBRATION_UPDATE event to be notified when valid calibration data has been found. It may also subscribe to the HL_EVENT_CALIBRATION_INPUT event when to be notified when the device requires user input (such as inserting the stylus into the inkwell) in order to obtain valid data. Once valid calibration has been found, hlUpdateCalibration is called to flush that data to the haptic device. Note that the change in calibration may cause a large discontinuity in the positions reported by the device. For this reason, hlUpdateCalibration should not be called while the user in the middle of an operation that relies on continuity of reported device positions (for example, drawing or manipulating an object). Users may also subscribe the to HL_CALIBRATION_UPDATE

Example:

```
void HLCALLBACK calibrationCallback(HLenum event,
                                        HLuint object,
                                        HLenum thread,
                                    HLcache *cache,
                                        void *userdata)
{
    if (event == HL_EVENT_CALIBRATION_UPDATE)
    {
          std::cout << "Device requires calibration update..." << std::endl;
          hlUpdateCalibration();
    }
    else if (event == HL_EVENT_CALIBRATION_INPUT)
    {
          std::cout << "Device requires calibration.input..." << std::endl;
    }
}

…

hlAddEventCallback(HL_EVENT_CALIBRATION_UPDATE, HL_OBJECT_ANY,
                     HL_CLIENT_THREAD, &calibrationCallback, NULL);
hlAddEventCallback(HL_EVENT_CALIBRATION_INPUT, HL_OBJECT_ANY,
                     HL_CLIENT_THREAD, &calibrationCallback, NULL);
```

Errors:

- HL_INVALID_ENUM if pname is not one of the values listed above.
- HL_INVALID_VALUE if value is out of range.
- HL_INVALID_OPERATION if no haptic rendering context is current.

See also: hlAddEventCallback

Appendix A Haptic Device API Parameters

The following pages contain tables that list the names of the HDAPI parameters. For each parameter the table lists a description and what types and how many values are used by each parameter name. The developer is responsible for supplying the correct number of parameters for the particular parameter name. Not all parameter names support every type.

The parameters are grouped as follows:

Topic	**Page**
Get Parameters	A-2
Set Parameters	A-8
Capability Parameters	A-10
Codes	A-11

Get Parameters

Table A-1: hdGet Parameters

Parameter Name	Description	Number of Arguments	Allowed Types	Units
HD_CURRENT_BUTTONS	Get the button state. Individual button values can be retrieved by doing bitwise & with HD_DEVICE_BUTTON_N (N=1,2,3, or 4).	1	HDint	
HD_CURRENT_SAFETY_SWITCH	Get whether the safety switch, if one exists, is active. For example, the PHANTOM Desktop's safety switch is true if the conductive portion of the stylus is being held.	1	HDboolean	
HD_CURRENT_ENCODER_VALUES	Get the raw encoder values.	6	HDlong	
HD_CURRENT_POSITION	Get the current position of the device facing the device base. Right is + x, up is +y, toward user is +z.	3	HDdouble, HDfloat	mm
HD_CURRENT_VELOCITY	Get the current velocity of the device. Note: This value is smoothed to reduce high frequency jitter.	3	HDdouble, HDfloat	mm/s
HD_CURRENT_TRANSFORM	Get the row-major transform of the device end-effector.	16	HDdouble, HDfloat	
HD_CURRENT_ANGULAR_VELOCITY	Gets the angular velocity of the device gimbal.	3	HDdouble, HDfloat	rad/s
HD_CURRENT_JOINT_ANGLES	Get the joint angles of the device. These are joint angles used for computing the kinematics of the armature relative to the base frame of the device. For PHANTOM devices: Turet Left +, Thigh Up +, Shin Up +	3	HDdouble, HDfloat	rad
HD_CURRENT_GIMBAL_ANGLES	Get the angles of the device gimbal. For PHANTOM devices: From Neutral position Right is +, Up is -, CW is +	3	HDdouble, HDfloat	rad

Table A-2: Get Forces Parameters

Parameter Name	**Description**	**Number of Arguments**	**Allowed Types**	**Units**
HD_CURRENT_FORCE	Get the current force, i.e. the force that the user is commanding to the device during the frame in which this is called. Returns 0 if no force has been commanded yet in the frame.	3	HDfloat, HDdouble	N
HD_CURRENT_TORQUE	Get the current torque, i.e. the torque that the user is commanding to the device during the frame in which this is called. Returns 0 if no torque has been commanded yet in the frame.	3	HDfloat, HDdouble	Nm
HD_CURRENT_MOTOR_DAC_VALUES	Get the current motor DAC, i.e. the motor value that the user is commanding to the device during the frame in which this is called. Returns 0 if no DAC has been commanded yet in the frame.	3 or 6	HDlong	range of "-32768 to 32767"

Table A-3: Get Identification Parameters

Parameter Name	Description	Number of Arguments	Allowed Types	Units
HD_HARDWAVE_REVISION	Get the hardware revision.	1	HDdouble	
HD_VERSION	Get the HDAPI software version, in the form of major.minor.build. This is taken from the HD_ VERSION_ definitions.	1	HDstring	
HD_DEVICE_MODEL_TYPE	Get a readable string of the device model type	1	HDstring	
HD_DEVICE_DRIVER_VERSION	Get a readable string of the driver version.	1	HDstring	
HD_DEVICE_VENDOR	Get a readable string of the device vendor.	1	HDstring	
HD_DEVICE_SERIAL_NUMBER	Gets a readable string of the device serial number.	1	HDstring	
HD_MAX_WORKSPACE_DIMENSIONS	Get the maximum workspace dimensions of the device, i.e. the maximum mechanical limits of the device, as (minX, minY, minZ, maxX, maxY, maxZ)	6	HDfloat, HDdouble	mm
HD_USABLE_WORKSPACE_DIMENSIONS	Get the usable workspace dimensions of the device, i.e. the practical limits for the device, as (minX, minY, minZ, maxX, maxY, maxZ). It is guaranteed that forces can be reliably rendered within the usable workspace dimensions.	6	HDfloat, HDdouble	mm
HD_TABLETOP_OFFSET	Get the mechanical offset of the device end-effector in Y from the table top.	1	HDfloat	mm
HD_INPUT_DOF	Get the number of input degrees of freedom. (i.e. the number of independent position variables needed to fully specify the end-effector location for PHANTOM device) 3DOF input means xyz tranlsational sensing-only. 6DOF means 3 translation abd 3 rotation.	1	HDint	

Table A-3: Get Identification Parameters

HD_OUTPUT_DOF	Get the number of output degrees of freedom, i.e. the number of independent actuation variable. For PHANTOM devices 3DOF means XYZ linear force output whereas 6DOF means xyz linear forces and roll, pitch, yaw, torques about gimbal.	1	HDint	
HD_CALIBRATION_STYLE	The style(s) of calibration supported by the device. Can be one or more bitwise of the following HD_CALIBRATION_AUTO, HD_ CALIBRATION_ENCODER_RESET, or HD_ CALIBRATION_INKWELL	1	HDint	

Table A-4: Get Last Values Parameters

Parameter Name	Description	Number of Arguments	Allowed Types	Units
HD_LAST_BUTTONS	Get last frame's button state. Individual button values can be retrieved by doing bitwise & with HD_DEVICE_BUTTON_N (N=1,2,3, or 4).	1	HDint	
HD_LAST_SAFETY_SWITCH	Get last frame's safety switch active status.	1	HDboolean	
HD_LAST_ENCODER_VALUES	Get last frame's raw encoder values.	6	HDlong	
HD_LAST_POSITION	Get the last position of the device facing the device base. Right is + x, up is +y, toward user is +z.	3	HDdouble, HDfloat	mm
HD_LAST_VELOCITY	Get the last velocity of the device. Note: this value is smoothed to reduce high frequency jitter.	3	HDdouble, HDfloat	mm/s
HD_LAST_TRANSFORM	Get the last row-major transform of the device end-effector.	16	HDdouble, HDfloat	
HD_LAST_ANGULAR_VELOCITY	Get the last Cartesian angular velocity of the device gimbal.	3	HDdouble, HDfloat	rad

Table A-4: Get Last Values Parameters

HD_LAST_JOINT_ANGLES	Get the last joint angles of the device. These are joint angles used for computing the kinematics of the armature relative to the base frame of the device. For PHANTOM devices: Turet Left +, Thigh Up +, Shin Up +	3	HDdouble, HDfloat	rad
HD_LAST_GIMBAL_ANGLES	Get the last angles of the device gimbal. For PHANTOM devices: From Neutral position Right is +, Up is -, CW is +	3	HDdouble, HDfloat	rad/s

Table A-5: Get Safety Parameters

Parameter Name	Description	Number of Arguments	Allowed Types	Units
HD_NOMINAL_MAX_STIFFNESS	Get the maximum closed loop stiffness that is recommended for the device, i.e. the spring constant used in F=kx output where k is the spring constant and x is the spring length.	1	HDfloat, HDdouble	0 to 1
HD_NOMINAL_MAX_DAMPING	Get the maximum level of damping that is recommended for the device, i.e. the damping constant used in F=kx-dv where d is the damping constant and v is the velocity of the device.	1	HDfloat, HD Double	0 to 1
HD_NOMINAL_MAX_FORCE	Get the nominal maximum force, i.e. the amount of force that the device can sustain when the motors are at room temperature (optimal).	1	HDfloat, HDdouble	N
HD_NOMINAL_MAX_CONTINUOUS_FORCE	Get the nominal maximum continuous force, i.e. the amount of force that the device can sustain through a period of time.	1	HDfloat, HDdouble	N
HD_MOTOR_TEMPERATURE	Get the motor temperature, which is the predicted temperature of all of the motors.	3 or 6	HDfloat, HDdouble	0(coldest) to 1(warmest)
HD_SOFTWARE_MAX_VELOCITY	Get the software maximum velocity limit. This does not replace the hardware limit of the device.	1	HDfloat, HDdouble	mm/s

Table A-5: Get Safety Parameters

HD_FORCE_RAMPING_RATE	Get the force ramping rate, which is the rate that the device ramps up forces when the scheduler is started or after an error.	1	HDfloat, HDdouble	N/s

Table A-6: Get Scheduler Update Codes Parameters

Parameter Name	Description	Number of Arguments	Allowed Types	Units
HD_UPDATE_RATE	Get the average update rate of the device, i.e. the number of updates that the scheduler performs per second.	1	HDint	Hz
HD_INSTANTANEOUS_UPDATE_RATE	Get the instantaneous update rate of the device, i.e. I/T where T is the time in seconds since the last update.	1	HDint	Hz

Set Parameters

Table A-7: Set Forces Parameters

Parameter Name	Description	Number of Arguments	Allowed Types	Units
HD_CURRENT_FORCE	Set the current force as Cartesian coordinated vector. This is the primary method for sending forces to the device. Setting the current force causes that force to be commanded at end of the frame.	3	HDfloat, HDdouble	N
HD_CURRENT_TORQUE	Set the current torque Cartesian coordinated vector, for 6DOF devices This is the primary method for sending torques to the device. Setting the current torque causes that torque to be commanded at end of the frame.	3	HDfloat, HDdouble	Nm
HD_CURRENT_MOTOR_DAC_VALUES	Set the current motor DAC values. THis is the primary method for commanding the amount of motor torque counts. This method cannot presently be used in combination with Cartesian force or torque commands.	3 or 6	HDlong	range of "-32768 to 32767"

Table A-8: Set Safety Parameters

Parameter Name	Description	Number of Arguments	Allowed Types	Units
HD_SOFTWARE_VELOCITY_LIMIT	Set the software maximum velocity limit. This does not replace the hardware limit of the device.	1	HDfloat, HDdouble	mm/s
HD_SOFTWARE_FORCE_IMPULSE_LIMIT	Sets the software maximum force impulse limit, which is the maximum change in magnitude and direction calculated by taking the difference between the current and last commanded force.	1	HDfloat, HDdouble	N/ms
HD_FORCE_RAMPING_RATE	Set the force ramping rate, which is the rate that the device ramps up forces when the scheduler is started or after an error.	1	HDfloat, HDdouble	N/s

Capability Parameters

Table A-9: hdEnable, hdDisable Parameters

Parameter Name	Description
HD_FORCE_OUTPUT	Enables or disables force output for the device. All motors are turned on or off.
HD_MAX_FORCE_CLAMPING	Enables or disables max force clamping for the device. If enabled, this will clamp the overall force output magnitude to the maxi achievable.
HD_FORCE_RAMPING	Enables or disables force ramping, i.e. whether the device ramps up forces when the scheduler is turned on.
HD_SOFTWARE_FORCE_LIMIT	Enables or disables the software maximum force check, which returns an error if the force magnitude commanded to the device exceeds the maximum force limit. This is primarily to disable force kicking. Disable this at your own risk.
HD_SOFTWARE_FORCE_IMPULSE_LIMIT	Enables or disables the software maximum force impulse check, which prevents changes in force impulse and direction that exceed the change of impulse limit. This is primarily to prevent device kicking. Disable this at your own risk.
HD_SOFTWARE_VELOCITY_LIMIT	Enables or disables the software maximum velocity check, which returns an error if the velocity magnitude commanded to the device exceeds the maximum velocity limit. This is primarily to disable force kicking. Disable this at your own risk.
HD_ONE_FRAME_LIMIT	Enables or disables the one frame limit check, which restricts the application to one haptics frame per scheduler tick This should only be disabled if the developer is running his own scheduler.

Codes

Table A-10: Calibration Return Codes

Parameter Name	Description
HD_CALIBRATION_OK	Calibration is accurate.
HD_CALIBRATION_NEEDS_UPDATE	Calibration needs an update. Call hdUpdateCalibration to have the device update its calibration.
HD_CALIBRATION_NEEDS_MANUAL_INPUT	Calibration needs manual input, such as having the user put the device in a reset position or inkwell. Once the user does this, further calls should no longer return that the calibration needs manual input.

Table A-11: Calibration Styles

Parameter Name	Description
HD_CALIBRATION_AUTO	The device will gather new calibration information as the armature is moved.
HD_CALIBRATION_ENCODER_RESET	The device needs to be put in a reset position to be calibrated.
HD_CALIBRATION_INKWELL	Inkwell calibration. The device needs to put into a fixture, i.e. inkwell, before calibration can be performed.

Table A-12: Device Button Codes

Parameter Name	Description
HD_DEVICE_BUTTON_N (N=1,2,3, or 4)	Button states for current and last buttons. Use bitwise & to extract individual button information.

Table A-13: Scheduler Priority Codes

Parameter Name	Description
HD_MAX_SCHEDULER_PRIORITY	Maximum priority for a scheduler callback. Setting a callback with this priority means that the callback will be run first every scheduler tick in relation to other callbacks.
HD_MIN_SCHEDULER_PRIORITY	Minimum priority for a scheduler callback. Setting a callback with this priority means that the callback will be run last every scheduler tick in relation to other callbacks.
HD_DEFAULT_SCHEDULER_PRIORITY	Default scheduler priority. Set this for callbacks that do not care about when they are executed during the scheduler tick. This value is between min and max priority.

Appendix B Haptic Library API Parameters

The following pages contain tables that list the names of the HLAPI parameters. For each parameter the table lists a description.

The parameters are grouped as follows:

State Maintenance Parameters

Table B-1: hlGetBooleanv, hlGetDoublev

Parameter Name	Description
HL_BUTTON1_STATE	Single Boolean value representing the first button on the haptic device. A value of true means that the button is depressed.
HL_BUTTON2_STATE	Single Boolean value representing the second button on the haptic device. A value of true means that the button is depressed.
HL_PROXY_IS_TOUCHING	Single Boolean value set to true when the proxy is in contact with one or shapes.
HL_PROXY_TOUCH_NORMAL	A vector of 3 doubles representing the surface normal at the point of contact with the set of shapes in contact with the proxy. Only valid if HL_PROXY_ IS_TOUCHING is true.
HL_PROXY_POSITION	Vector of 3 doubles representing the position of the proxy in world coordinates.
HL_PROXY_ROTATION	Vector of 4 doubles representing a quaternion that specifies the rotation of the proxy in world coordinates.
HL_PROXY_TRANSFORM	Vector of 16 doubles representing a 4x4 transform matrix in column major order that specifies the transform of the proxy relative to world coordinates
HL_DEVICE_POSITION	Vector of 3 doubles representing the position of the haptic device in world coordinates.
HL_DEVICE_ROTATION	Vector of 4 doubles representing a quaternion that specifies the rotation of the haptic device in world coordinates.
HL_DEVICE_TRANSFORM	Vector of 16 doubles representing a 4x4 transform matrix in column major order that specifies the transform of the haptic device relative to world coordinates.
HL_DEVICE_FORCE	Vector of 3 doubles representing the last force, in world coordinates, sent to the haptic device.
HL_DEVICE_TORQUE	Vector of 3 doubles representing the last torque, in world coordinates, sent to the haptic device.

Table B-1: hlGetBooleanv, hlGetDoublev

HL_EVENT_MOTION_LINEAR_TOLERANCE	Double precision value representing the minimum distance, in device workspace coordinates, that the proxy must move before a motion event is triggered.
HL_EVENT_MOTION_ANGULAR_TOLERANCE	Double precision value representing the minimum rotation, in radians, that the proxy must move before a motion event is triggered.
HL_VIEWTOUCH_MATRIX	Array of sixteen doubles representing a 4x4 transform matrix in column major order that specifies the transform from view coordinates to touch coordinates.
HL_TOUCHWORKSPACE_MATRIX	Array of sixteen doubles representing a 4x4 transform matrix in column major order that specifies the transform from touch coordinates to workspace coordinates.

TableB-2: hlCacheGetBooleanv, hlCacheGetDoublev

Parameter Name	Description
HL_BUTTON1_STATE	Single Boolean value representing the first button on the haptic device. A value of true means that the button is depressed.
HL_BUTTON2_STATE	Single Boolean value representing the second button on the haptic device. A value of true means that the button is depressed.
HL_PROXY_IS_TOUCHING	Single Boolean value set to true when the proxy is in contact with one or shapes.
HL_PROXY_TOUCH_NORMAL	A vector of 3 doubles representing the surface normal at the point of contact with the set of shapes in contact with the proxy. Only valid if HL_PROXY_IS_TOUCHING is true.
HL_PROXY_POSITION	Vector of 3 doubles representing the position of the proxy in world coordinates.
HL_PROXY_ROTATION	Vector of 4 doubles representing a quaternion that specifies the rotation of the proxy in world coordinates.
HL_PROXY_TRANSFORM	Vector of 16 doubles representing a 4x4 transform matrix in column major order that specifies the transform of the proxy relative to world coordinates.
HL_DEVICE_POSITION	Vector of 3 doubles representing the position of the haptic device in world coordinates.
HL_DEVICE_ROTATION	Vector of 4 doubles representing a quaternion that specifies the rotation of the haptic device in world coordinates.
HL_DEVICE_TRANSFORM	Vector of 16 doubles representing a 4x4 transform matrix in column major order that specifies the transform of the haptic device relative to world coordinates.
HL_DEVICE_FORCE	Vector of 3 doubles representing the last force, in world coordinates, sent to the haptic device.
HL_DEVICE_TORQUE	Vector of 3 doubles representing the last torque, in world coordinates, sent to the haptic device.

TableB-3: hlGet Error

Parameter Name	Description
HL_NO_ERROR	No error. The last call API function call was successful.
HL_INVALID_ENUM	An invalid valid for a enumerated type was passed to an API function. The function call will have no effect other than to set the error.
HL_INVALID_VALUE	A value passed to an API function is outside the valid range for that function. The function call will have no effect other than to set the error.
HL_INVALID_OPERATION	An API function was called when the renderer was not in an appropriate state for that function call. For example, hlBeginShape was called outside an hlBeginFrame/hlEndFrame pair, or hlBeginFrame when no haptic rendering context was active. The function call will have no effect other than to set the error.
HL_STACK_OVERFLOW	An API function was called that would have caused an overflow of the matrix stack. The function call will have no effect other than to set the error.
HL_STACK_UNDERFLOW	An API function was called that would have caused an underflow of the matrix stack, i.e. a call to hlPopMatrix when the stack is empty. The function call will have no effect other than to set the error.
HL_OUT_OF_MEMORY	There is not enough memory to complete the last API function called. This function may have partially completed leaving the haptic renderer in an undefined state.
HL_DEVICE_ERROR	An error occurred with the haptic device. The errorInfo field of the HLerror struct will contain a device specific error code. See the HDAPI reference guide for a list of device errors. This error may be signaled even outside any API function calls if the device error is detected in the haptic rendering engine running in a separate thread. The haptic device recovers in most cases.

TableB-4: hlGetString

Parameter Name	Description
HL_VENDOR	Returns the name of the company responsible for the haptic renderer implementation.
HL_VERSION	Returns the version number of the haptic rendering library as a string of the form: major_number.minor_number.build_number

TableB-5: hlHinti, hlHintb

Parameter Name	Description
HL_SHAPE_FEEDBACK_BUFFER_VERTICES	A single integer value indicating to the haptic rendering engine approximately how many vertices will be in the next feedback buffer shape so that it may reserve the correct amount of memory.
HL_SHAPE_DYNAMIC_SURFACE_CHANGE	A single Boolean value indicating whether or not shapes that follow should be treated as dynamically changing surfaces. A value of true tells the haptic rendering engine to do extra processing to ensure that the proxy does not fall through a dynamically changing surface. A value of false tells the rendering engine to optimize haptic rendering for surfaces which will not change.

Shape Parameters

TableB-6: hlBegin Shape

Parameter Name	**Description**
HL_SHAPE_FEEDBACK_BUFFER	OpenGL commands used following the hlBeginShape call will generate a set of geometric primitives (quads, triangles, points and lines) that will be sent to the OpenGL feedback buffer. When hlEndShape is called, these primitives will be read out of the feedback buffer and sent to the haptic renderer. Feedback buffer shapes can be used to specify triangles meshes as well as points and lines for constraints. For optimal memory usage and performance, use hlHint with HL_SHAPE_FEEDBACK_BUFFER_VERTICES before hlBeginShape, so that the size of the feedback buffer may be optimally allocated.
HL_SHAPE_DEPTH_BUFFER	OpenGL commands used following the hlBeginShape call will generate an image in the OpenGL depth buffer. When hlEndShape is called, this image will be read out of the depth buffer and sent to the haptic renderer. Any OpenGL commands that modify the depth buffer may be used. Point and line constraints may not be specified using a depth buffer shape.
HL_SHAPE_CALLBACK	Allows clients to create custom shapes not supported using the other shape types. OpenGL commands are ignored for this shape type. No geometry is specified. Instead, the current shape intersection and shape closest point callbacks are used for haptic rendering of this shape.

TableB-7: hlShapeGetBooleanv, hlShapeGetDoublev

Parameter Name	**Description**
HL_PROXY_IS_TOUCHING	Single Boolean value. True means that the proxy is currently in contact with the shape.
HL_REACTION_FORCE	Vector of 3 doubles representing the contribution of this shape to the overall reaction force that was sent to the haptic device during the last frame. If the proxy was not in contact with this shape last frame, this vector will be zero.

TableB-8: hlLocalFeature types

Type Name	**Description**
HL_LOCAL_FEATURE_POINT	The local feature is a single point whose position is given by the vector v.
HL_LOCAL_FEATURE_LINE	The local feature is a line segment whose start point and end point are given by the vectors v1 and v2 respectively.
HL_LOCAL_FEATURE_PLANE	The local feature is a plane whose normal is given by the vector v1 and that passes through the point v1.

Capability Parameters

TableB-9: hlEnable, hlDisable, hlIsEnabled

Parameter Name	Description
HL_PROXY_RESOLUTION	If enabled, use shape geometry to automatically update the proxy position. The computed proxy position will be valid for all shapes specified each frame. If disabled, shapes will be ignored and the proxy position will be set by the client program. This is enabled by default
HL_HAPTIC_CAMERA_VIEW	Enhances shape rendering by modifying the shape viewing transform based on the motion of the proxy. This allows the user to feel depth buffer features that are occluded in the primary camera view. The haptic camera view also serves as an optimization by sizing the viewing volume and viewport so that the graphics pipeline only processes geometry that is within a proximity to the proxy. This is disabled by default.
HL_ADAPTIVE_VIEWPORT	If enabled, this feature serves as an optimization for single pass rendering of depth buffer shapes. This feature adaptively sizes the read-back pixel dimensions of the viewport based on the location and motion of the proxy. This feature is incompatible with haptic camera view. This is disabled by default

Material and Surface Parameters

TableB-10: hlMaterialf - face values

Parameter Name	Description
HL_FRONT	Apply the material property only to the front face of shapes and to all faces of constraints.
HL_BACK	Apply the material property only to the back face of shapes and to all faces of constraints.
HL_FRONT_AND_BACK	Apply the material property to both front and back faces.

TableB-11: hlMaterialf - pname values

Parameter Name	Description
HL_STIFFNESS	Stiffness controls how hard surfaces feel. Param must be a value between 0 and 1 where 1 represents the hardest surface the haptic device is capable of rendering and 0 represents the most compliant surface that can be rendered.
HL_DAMPING	Damping reduces the springiness of the surface. Param must be between 0 and 1 where 0 represents no damping, i.e. a highly springy surface and 1 represents the maximum level of damping possible
HL_STATIC_FRICTION	Static friction controls the resistance of a surface to tangential motion when the proxy position is not changing i.e. how hard it is to slide along the surface starting from a complete stop. A param value of 0 is a completely frictionless surface and a value of 1 is the maximum amount of static friction the haptic device is capable of rendering.
HL_DYNAMIC_FRICTION	Dynamic friction controls the resistance of a surface to tangential motion when the proxy position is changing i.e. how hard it is to slide along the surface once already moving. A param value of 0 is a completely frictionless surface and a value of 1 is the maximum amount of dynamic friction the haptic device is capable of rendering.

TableB-12: hlGetMaterialfv - face values

Parameter Name	Description
HL_FRONT	Query the material property of the front face of shapes and all constraints.
HL_BACK	Query the material property of the back face of shapes and all constraints.

TableB-13: hlGetMaterialfv - pname values

Parameter Name	Description
HL_STIFFNESS	Stiffness controls how hard surfaces feel. Param must be a value between 0 and 1 where 1 represents the hardest surface the haptic device is capable of rendering and 0 represents the most compliant surface that can be rendered.
HL_DAMPING	Damping reduces the springiness of the surface. Param must be between 0 and 1 where 0 represents no damping, i.e. a highly springy surface and 1 represents the maximum level of damping possible.

Parameter Name	**Description**
HL_STATIC_FRICTION	Static friction controls the resistance of a surface to tangential motion when the proxy position is not changing i.e. how hard it is to slide along the surface starting from a complete stop. A param value of 0 is a completely frictionless surface and a value of 1 is the maximum amount of static friction the haptic device is capable of rendering.
HL_DYNAMIC_FRICTION	Dynamic friction controls the resistance of a surface to tangential motion when the proxy position is changing i.e. how hard it is to slide along the surface once already moving. A param value of 0 is a completely frictionless surface and a value of 1 is the maximum amount of dynamic friction the haptic device is capable of rendering.

TableB-14: hlTouchableFace - mode values

Parameter Name	**Description**
HL_FRONT	Only front faces will be touchable.
HL_BACK	Only back faces will be touchable.
HL_FRONT_AND_BACK	All faces, both front and back, will be touchable.

TableB-15: hlTouchModel

Parameter Name	**Description**
HL_CONTACT	The proxy position is not allowed to pass through geometric primitives (triangles). The proxy may move off the surface but it will remain on one side of it.
HL_CONSTRAINT	The proxy position is constrained to remain exactly on the surface of geometric primitives and it may only be moved off of the surface if the distance between the device position and the proxy is greater than the snap distance.

TableB-16: hlTouchModelIf

Parameter Name	**Description**
HL_SNAP_DISTANCE	Distance between the proxy position and the surface that must be exceeded to pull off a constraint. Param should be a floating point value representing the distance in millimeters in workspace coordinates. The default value is FLT_MAX to always be active.

Force Effect Parameters

TableB-17: hlStartEffect - effect types

Parameter Name	Description
HL_EFFECT_CONSTANT	Adds a constant force vector to the total force sent to the haptic device. The effect property HL_EFFECT_PROPERTY_DIRECTION specifies the direction of the force vector. The effect property HL_EFFECT_PROPERTY_MAGNITUDE specifies the magnitude of the force vector.
HL_EFFECT_SPRING	Adds a spring force to the total force sent to the haptic device. The spring force pulls the haptic device towards the effect position and is proportional to the product of the gain and the distance between the effect position and the device position. Specifically, the spring force is calculated using the expression F = k(P-X) where F is the spring force, P is the effect position, X is the current haptic device position and k is the gain. The effect position is specified by the property HL_EFFECT_PROPERTY_POSITION. The gain is specified by the property HL_EFFECT_PROPERTY_GAIN. The magnitude of the effect force is capped at the value of the effect property HL_EFFECT_MAGNITUDE.
HL_EFFECT_VISCOUS	Adds a viscous force to the total force sent to the haptic device. The viscous force is based on the current velocity of the haptic device and is calculated to resist the motion of the haptic device. Specifically the force is calculated using the expression F = -kV where f is the spring force, V is the velocity and k is the gain. The gain is specified by the property HL_ EFFECT_PROPERTY_GAIN. The magnitude of the effect force is capped at the value of the effect property HL_EFFECT_MAGNITUDE.
HL_EFFECT_FRICTION	Adds a friction force to the total force sent to the haptic device. Unlike friction specified via hlMaterial calls, this is friction both while touching objects and in free space. The gain of the friction force is specified by the property HL_EFFECT_PROPERTY_GAIN. The magnitude of the effect force is capped at the value of the effect property HL_EFFECT_ MAGNITUDE.
HL_EFFECT_CALLBACK	Allows for the user to create a custom effect by setting effect callbacks using hlCallback

TableB-18: hlTriggerEffect

Parameter Name	Description
HL_EFFECT_CONSTANT	Adds a constant force vector to the total force sent to the haptic device. The effect property HL_EFFECT_PROPERTY_DIRECTION specifies the direction of the force vector. The effect property HL_EFFECT_PROPERTY_MAGNITUDE specifies the magnitude of the force vector.
HL_EFFECT_SPRING	Adds a spring force to the total force sent to the haptic device. The spring force pulls the haptic device towards the effect position and is proportional to the product of the gain and the distance between the effect position and the device position. Specifically, the spring force is calculated using the expression F = k(P-X) where F is the spring force, P is the effect position, X is the current haptic device position and k is the gain. The effect position is specified by the property HL_EFFECT_PROPERTY_POSITION. The gain is specified by the property HL_EFFECT_PROPERTY_GAIN. The magnitude of the effect force is capped at the value of the effect property HL_EFFECT_MAGNITUDE.
HL_EFFECT_VISCOUS	Adds a viscous force to the total force sent to the haptic device. The viscous force is based on the current velocity of the haptic device and is calculated to resist the motion of the haptic device. Specifically the force is calculated using the expression F = -kV where f is the spring force, V is the velocity and k is the gain. The gain is specified by the property HL_EFFECT_PROPERTY_GAIN. The magnitude of the effect force is capped at the value of the effect property HL_EFFECT_MAGNITUDE.
HL_EFFECT_FRICTION	Adds a friction force to the total force sent to the haptic device. Unlike friction specified via hlMaterial calls, this is friction both while touching objects and in free space. The gain of the friction force is specified by the property HL_EFFECT_PROPERTY_GAIN. The magnitude of the effect force is capped at the value of the effect property HL_EFFECT_MAGNITUDE.
HL_EFFECT_CALLBACK	Allows for the user to create a custom effect by setting effect callbacks using hlCallback.

TableB-19: hlEffectd, hlEffecti, hlEffectdv, hlEffectiv

Parameter Name	Description
HL_EFFECT_PROPERTY_GAIN	Used by spring, friction and viscous effect types. Higher gains will cause these effects to generate larger forces.
HL_EFFECT_PROPERTY_MAGNITUDE	Used by constant, spring, friction and viscous effect types. Represents a cap on the maximum force generated by these effects.

Parameter Name	Description
HL_EFFECT_PROPERTY_FREQUENCY	Reserved for use by future effect types and callback effects.
HL_EFFECT_PROPERTY_DURATION	Used for all effects started with a call to hlTriggerEffect. The effect will automatically be terminated when the duration has elapsed. Duration is specified in milliseconds.
HL_EFFECT_PROPERTY_POSITION	Used by spring effect. Represents the anchor position of the spring.
HL_EFFECT_PROPERTY_DIRECTION	Used by constant effect. Represents direction of constant force vector.

TableB-20: hlGetEffectdv, hlGetEffectiv

Parameter Name	Description
HL_EFFECT_PROPERTY_GAIN	Used by spring, friction and viscous effect types. Higher gains will cause these effects to generate larger forces.
HL_EFFECT_PROPERTY_MAGNITUDE	Used by constant, spring, friction and viscous effect types. Represents a cap on the maximum force generated by these effects.
HL_EFFECT_PROPERTY_FREQUENCY	Reserved for use by future effect types and callback effects.
HL_EFFECT_PROPERTY_DURATION	Used for all effects started with a call to hlTriggerEffect. The effect will automatically be terminated when the duration has elapsed. Duration is specified in milliseconds.
HL_EFFECT_PROPERTY_POSITION	Used by spring effect. Represents the anchor position of the spring.
HL_EFFECT_PROPERTY_DIRECTION	Used by constant effect. Represents direction of constant force vector.

Callback Parameters

TableB-21: hlCallback

Parameter Name	Description
HL_SHAPE_INTERSECT_LS	Callback to intersect a line segment with a user defined custom shape. The callback function must have the following signature: `HLboolean HLCALLBACK fn(const HLdouble startPt[3], const HLdouble endPt[3], HLdouble intersectionPt[3], HLdouble intersectionNormal[3], void *userdata)` Where input parameters startPt and endPt are the endpoints of a line segment to intersect with the custom shape, intersectionPt is used to return the point of intersection of the line segment with the shape. In the case of multiple intersections, intersectionPt should be the closest intersection to startPt. IntersectionNormal is the surface normal of the custom shape at intersectionPt. Userdata is the same userdata pointer passed to hlCallback. The callback function should return true if the line segment intersects the shape. All data is in the local coordinate system of the custom shape.
HL_SHAPE_CLOSEST_FEATURE	Callback to find the closest point on the surface to an input point as well as one ore more local features that approximate the surface in the vicinity of that point. The callback function must have the following signature: `HLboolean HLCALLBACK fn(const HLdouble queryPt[3], const HLdouble targetPt[3], HLgeom *geom, HLdouble closestPt[3], void* userdata)` Where closest point should be set to the closest point on the surface of the custom shape closest to the input parameter queryPt. Normal is the surface normal of the custom shape at closestPt. Userdata is the same userdata pointer passed to hlCallback. All data is in the local coordinate system of the custom shape.

Event Parameters

TableB-22: hlAddEventCallback, hlRemoveEventCallBack - event values:

Parameter Name	Description
HL_EVENT_MOTION	Callback will be called when either the proxy has moved more than the HL_EVENT_MOTION_LINEAR_TOLERANCE millimeters in workspace coordinates from the position when the last motion event was triggered or when the proxy has been rotated more than HL_EVENT_MOTION_ANGULAR_TOLERANCE radians from the rotation of the proxy last time a motion event was triggered.
HL_EVENT_1BUTTONDOWN	Callback will be called when the first button on the haptic device is depressed.
HL_EVENT_1BUTTONUP	Callback will be called when the first button on the haptic device is released.
HL_EVENT_2BUTTONDOWN	Callback will be called when the second button on the haptic device is depressed.
HL_EVENT_2BUTTONDOWN	Callback will be called when the second button on the haptic device is released.
HL_EVENT_TOUCH	Callback will be called when a shape in the scene has been touched (the proxy is in contact with the shape).
HL_EVENT_UNTOUCH	Callback will be called when the shape in the scene is no longer being touched (the proxy was in contact with shape but is no longer in contact).

TableB-23: hlAddEventCallback, hlRemoveEventCallback - thread values

Parameter Name	Description
HL_CLIENT_THREAD	Callback function will be called from client thread, when the client program calls hlCheckEvents.
HL_COLLISION_THREAD	Callback function will be called from the internal collision thread running in the haptic rendering engine. Most event callbacks should be handled in the client thread, however there are some cases where collision thread callbacks are needed due to timing requirements.

TableB-24: hlEventd - pname values

Parameter Name	Description
HL_EVENT_MOTION_LINEAR_TOLERANCE	Sets the minimum distance in device workspace coordinates, that the linear translation of the proxy must change before a motion event is triggered. By default this value is one millimeter
HL_EVENT_MOTION_ANGULAR_TOLERANCE	Sets the minimum angular distance in, that orientation of the proxy must change before a motion event is triggered. By default this value is 0.02 radians.

Proxy Parameters

TableB-25: hlProxydv

Parameter Name	Description
HL_PROXY_POSITION.	Sets the position of the proxy in world coordinates. If proxy resolution is enabled, this call will have no effect

Transform Parameters

TableB-26: hlMatrix - mode values

Parameter Name	Description
HL_VIEWTOUCH	All matrix manipulation commands will target the transform from view coordinates to touch coordinates.
HL_TOUCHWORKSPACE	All matrix manipulation commands will target the transform from touch coordinates to the local coordinates of the haptic device.

Index