HapticMASTER
Programming Reference Manual

MOOG B.V.	Visit/shipping address	Phone	+31 (252) 462 065
P.O. Box 187	Pesetaweg 53	fax	+31 (252) 462 001
2150 AD Nieuw-Vennep	2153 PJ Nieuw-Vennep	email	robotics@moog.com
The Netherlands.	The Netherlands.		http://www.moog.com

Contents

1 HapticAPI commands

1.1 HapticAPI.h main functions

The HapticAPI has the following functions that are explained in the following paragraphs:

- haDeviceOpen
- haDeviceClose
- haDeviceSendString
- haGetAPIDllVersion

There are also calls for handling the data logger. These data logger calls will be discussed in §6.2.

1.1.1 haDeviceOpen

The haDeviceOpen function allows a host computer to connect to a HapticMaster. The syntax of the function is as follows:

```
devID = haDeviceOpen ( inAddress )
```

Inputs:

`inAddress`	A **char array** with an IP4 address of the haptic device, which also can be predefined.

Outputs:

`devID`	A variable of type **long**, unique for each of the devices that are connected to the host computer. On failure to connect, the function will return -1.

An example of how to connect to a HapticMaster with IP address 10.30.211.1 is given below:

```
long dev1 = haDeviceOpen("10.30.211.1");
```

1.1.2 haDeviceClose

The haDeviceClose function allows a host computer to disconnect from a HapticMaster. The syntax of the function is the following:

```
returnValue = haDeviceClose ( devID )
```

Inputs:

`devID`	A variable of type **long** used as identifier for the device that is to be disconnected from the host computer.

Outputs:

`returnValue`	An **integer** that is -1 if closing the device failed (e.g. wrong devID supplied)

An example of how to disconnect from the HapticMaster that was connected in the previous example is given below:

```
int ret = haDeviceClose( dev1 );
```

1.1.3 haDeviceSendString

The haDeviceSendString function can be used to send commands to the HapticMaster in the form of a string. This function can be used to create effects and objects as well as set/get parameter values. The syntax of the function is as follows:

```
returnValue = haDeviceSendString (  devID,
                                    inCommand,
                                   outCommand )
```

Inputs:

	`devID`	A variable of type **long** used as identifier for the device (supplied as return value from function haDeviceOpen).
	`inCommand`	A **char array** containing the command that is to be passed to the HapticMaster. See §1.4 for details about format of limitations

Outputs:

	`returnValue`	An **integer** that is -1 if sending the command failed. 0 is returned if sending the command was successful (it still does not mean that the command executed successfully – see outCommand hereunder)
	`outCommand`	A **char array** containing the result of the command. If an error has occurred while **executing** the command, the returned char array will start with "`---ERROR: `" (see §0).

Examples of how to send commands to the haptic device with the *haDeviceSendString* are given below. The available commands that can be used in the string command are discussed in the following sections.

```
char res[200];
int ret;
ret = haDeviceSendString( devID, "create sphere mySphere",  res);
ret = haDeviceSendString( devID, "set mySphere pos [0.3,0.43,0.01]", res);
ret = haDeviceSendString( devID, "remove all", res);
```

1.1.4 haGetAPIDllVersion

The haGetAPIDllVersion function can be used to get the HapticAPI version of the HapticAPI.dll. The HapticAPI version of the dll can then be compared with the version running on the HapticMaster by using the command “`get api`” (see §2).

```
returnValue = haGetAPIDllVersion ( outRes )
```

Outputs:

`returnValue`	An **integer** that is -1 if fetching the DLL version failed
`outRes`	A **char array** containing the version of the Dll. The current version is 2.5.1.

1.2 Matlab

For interfacing the HapticMaster through Matlab we have 3 files with similarly named functions:

- haDeviceOpen.m
- haDeviceClose.m
- haDeviceSendString.m

The functions are to be found in the folder Matlab on the CD-ROM supplied with your device. They are in general similar to the ones discussed in §1.1. The difference will be in the interface. We will now discuss them briefly.

1.2.1 haDeviceOpen

```
function socket = haDeviceOpen ( host )
```

Inputs:

`host`	A string with an IP4 address of the haptic device, which also can be predefined.

Outputs:

`socket`	An identifier for the tcp/ip connection to the HapticMaster.

1.2.2 haDeviceClose

```
function haDeviceClose ( socket )
```

Inputs:

`socket`	An identifier for the tcp/ip connection to the

		HapticMaster (returned by haDeviceOpen on connection creation)

1.2.3 haDeviceSendString

```
function outRes = haDeviceSendString (   socket,
                                         command )
```

Inputs:

	`socket`	An identifier for the tcp/ip connection to the HapticMaster.
	`command`	A string containing the command that is to be passed to the haptic device.

Outputs:

	`outRes`	A string where the result of the command call is returned.

1.3 Telnet

It is possible to send commands to the HapticMaster using Telnet from any other computer of the same network. Upon connection, command strings can be used to access the functionality of the HapticMaster. The steps that need to be followed are (for Windows environment with telnet program):

a) Open a command prompt window (Windows → Run… → cmd)
b) Type ***`telnet IP_ADDRESS 7654`*** , where IP_ADDRESS is the IP address of the HapticMaster (e.g. ***`telnet 10.30.211.1 7654`***)
c) Type the desired string command (e.g. ***`create sphere mySphere1`***)
d) If there is an error, then an "`---ERROR: `" string will be printed in the command window (for more detail see §0), otherwise an affirmative response will be given (reporting success)
e) To close the connection, type the command ***`quit`***

1.4 Command Strings

Command strings are the commands that are passed to the HapticMaster. These can be supplied using the *inCommand* parameter of the *haDeviceSendString* function. A command string consists mostly of two parts: the action, and the property on which the action should be applied. In some cases the string command can only contain an action call like creating an effect.

The following four **actions** are available in the HapticMaster:

• ***set***	Set a property of the HapticMaster or effect
• ***get***	Get the current values of a property of the environment, effect, etc.
• ***create***	Create an effect in the HapticMaster's virtual world. Details on how to use this action are found in §3.1
• ***remove***	Remove an effect from the HapticMaster's virtual world. Details on how to use this action are found in §3.2

The property part of the command string for the *set* and the *get* actions consists of 1, 2 or 3 elements separated by a white space.

- A ***get*** action for a **global** property expects one parameter – namely the property itself
- A ***get*** action for an effect expects two parameters – the name of the effect and the property
- The ***set*** variant of the two actions described above expects an extra parameter, namely the value that you want to set to this property

The 4 options for the property part are described below:

<u>Please pay attention</u> to the fact that **exactly one white space** is expected between the elements of the command.

set *<globalProperty> <propertyValue>*
set *<effectName> <effectProperty> <propertyValue>*

For the ***get*** action the following overloads are available:

get *<globalProperty>*
get *<effectName> <effectProperty>*

where *<globalProperty>* is a ***property*** as shown in Table 2, *<effectName>* is the name of the effect whose property value is to be set / retrieved, *<effectProperty>* is a ***property*** as defined in §3. Examples of calls are:

```
get state
get myEffect pos
set state off
set myEffect pos [0.3,0.43,0.01]
```

When commanding a **set** action, the <*propertyValue(s)*> needs to be in one of the formats shown in **Error! Reference source not found.** depending on the property type. If we command a get action, the result will follow the similar formatting.

Type	***Format*** `Example`	***Description***
String	stringValue `DummyString`	A string of characters without any spaces. Strings are case sensitive
Boolean	boolValue `true`	A string stating true or false
Integer	intValue `3`	A string of characters depicting an integer value

Float	floatValue
	`3.456`
	`-0.4e-002`
		A string of characters depicting a float value. Exponent notation is also accepted

Array	[*val1*,*val2*,*val3*] `[0.01,0.012,1.325]`	A string of characters containing values separated by a comma ‘,’, no spaces, included by brackets []
Vector3d	[*x*,*y*,*z*] `[0.01,0.012,1.325]`	Same as Array, where x is the x component of the Vector3d, y is the y component and z is the z component
Quaternion	[*x*,*y*,*z*,*w*] `[0.747,0.0,0.0,0.747]`	Same as Array, where x is the x component of the quaternion, y is the y component, z is the z component, and w is the scalar component of the quaternion
Callback	stringValue `"set 'enable' succeeded"`	A string returning whether the call succeeded or failed

Table 1: *The format of the different property types*

1.5 Multiple string commands in one call

It is possible to send multiple commands in one single string. The commands are then separated with a semicolon ‘;’. The following example is a string command that gets the current state of the HapticMaster and sets the inertia to 3.0 kg. Command execution is sequential.

```
get state;set inertia 3.0
```

The results of this call will also be separated by a semicolon ‘;’.

```
	off;"inertia set";
```

We can concatenate more calls in one string. The only restriction here is that the HapticMaster does not accept strings longer than 2048 characters.

Error / Success strings

If an error occurs during execution of a command, an "`---ERROR: `" string will be printed in the response string followed by some extra information about the error. If the command execution succeeds and it was a ***set*** / ***create*** / ***remove*** command, a success report will be returned. Both the Error and the Success strings are **surrounded by the quote** characters (" ") the error.

By a successful ***get*** command execution, the returned string is **not surrounded by a quote** (e.g. `4.0` or `true` or `stop`).

Two examples are given below:

```
get help
"--- ERROR: Command 'get help' not found";
set force_calibrated true
"--- ERROR: force_calibrated is not writable";
set state stop
"state set";
```

2 Global HapticMASTER Properties

Global properties are the properties that are related to the HapticMaster End-Effector, and the ones that generally influence the behavior of the HapticMaster. The properties that are accessible through the API are listed in Table 2. The first column is the string that is used to access a certain global property. The second column is a description of the property. The third column is the variable type of the property (to be found in Table 1: ***The format of the different property types***). The fourth column shows the unit of the global parameter. Finally the fifth and sixth columns define if the property can be written (set) and/or read (get).

Property	Description	Type	Unit	Set	Get
emergencybuttonpushed	Is emergency button pushed?	Boolean	---	V	V
emergencyrelay	Is emergency relay on?	Boolean	---	V	V
inertia	End-Effector mass	Float	[Kg]	V	V
state	Current state	`off` `init` `home` `stop` `position` `force`	---	V	V
coulombfriction	Coulomb friction – minimum force applied before moving	Float	[N]	V	V
toggle_sync_bit	Toggle the sync bit of the LTP-port (clock)	Boolean	---	V	V
calibrateforcesensor	Calibrate the force sensor	Callback	---	V	
measpos	Measured position	Vector3D	[m]		V
measvel	Measured velocity	Vector3D	[m/s]		V
measforce	Measured force	Vector3D	[N]		V
modelpos	Model position	Vector3D	[m]		V
modelvel	Model velocity	Vector3D	[m/s]		V
modelacc	Model acceleration	Vector3D	[m/s^2]		V
measposjoint	Measured position joint axes	Vector3D	[m]		V
measforcejoint	Measured force joint axes	Array	[N]		V
modelposjoint	Model position joint axes	Array	[m]		V
modelveljoint	Model velocity joint axes	Array	[m/s]		V

modelaccjoint	Model acceleration joint axes	Array	[m/s^2]		V
pot1dir	Gimbal pot1 direction	Float	[rad]		V
pot2dir	Gimbal pot2 direction	Float	[rad]		V
pot3dir	Gimbal pot3 direction	Float	[rad]		V
force_calibrated	Is force calibrated?	Boolean	---		V
position_calibrated	Is position calibrated?	Boolean	---		V
workspace_r	R low stop, R distance of pivot to workspace origin, R high stop (see §4)	Array	[m]		V
workspace_phi	Phi low stop, 0, Phi high stop (see §4)	Array	[rad]		V
workspace_z	Z low stop, 0, Z high stop (see §4)	Array	[m]		V
version	Real-time version	String	---		V
api	HapticAPI version	String	---		V
os	Operating System	String	---		V
device	HapticMaster	String	---		V
date	Real-time build date	String	---		V

Table 2: *Description of the global properties*

Some example commands and their results:

```
set inertia 3.0
“inertia set”;
get state
off;
set state stop
“state set”;
get workspace_r
[-0.14,0.5,0.14];
```

Note that informative strings that are returned by a set statement and errors are returned from the API with leading and trailing quotes (“xxx”). All other results are returned without quotes.

3 Effects

3.1 Create effect

An effect can be created by supplying the following command:

create <*effectType*> <*effectName*>

effectType can be one of the following:

spring	- A spring
damper	- A damper
biasforce	- A bias force
shaker	- A shaker (oscillator)
block	- A solid block
sphere	- A solid sphere
flatplane	- A solid flat plane
cylinder	- A solid cylinder
capsule	- A solid capsule
torus	- A solid torus (donut)

effectName can be any string you want as long as it still does not exist in the haptic world and does not contain spaces.

Example commands:

```
create spring mySpring
"Effect spring with name mySpring created";
create sphere mySphere
"Effect sphere with name mySphere created";
```

3.2 Remove effect

Removing an effect can be done in two ways. By providing a name to remove a specific effect:

remove <*effectName*>

where *effectName* must be an existing effect.

Or we can delete all effects at once by calling:

remove all

Example commands:

```
remove mySpring
"Removed effect with name mySpring";
remove all
"Removed all effects";
```

3.3 General Effect properties

The common properties of the effects are the ones that generally describe the effect and are there regardless of the effect's type. The general properties of the effects are listed in Table 3.

Property	Description	Type	Unit	Set	Get
pos	Effect position	Vector3D	[m]	V	V
vel	Effect velocity	Vector3D	[m/s]	V	V
att	Effect attitude (orientation)	Quaternion	[x, y, z, w]	V	V
enable	Enables the effect	Callback	---	V	
disable	Disables the effect	Callback	---	V	
enabled	Is effect enabled?	Boolean	---		V

Table 3: *Description of the general effect properties*

Example commands:

```
set myBlock vel [0.0,0.0,0.5]
"vel set";
set mySpring enable
"set 'enable' succeeded";
get mySphere pos
[0.0,0.0,0.0];
set myTorus att [0,0.707,0,0.707]
“att set”
get myDamper enabled
true;
```

3.4 Surface properties (for Block, Sphere, Torus, Cylinder, Capsule, FlatPlane)

For all solid effects (block, sphere, torus, cylinder, capsule, flatplane) surface properties can be set / get. The properties apply for all surfaces of the specific effect and determine the behavior/feel of the object’s surface (see Table 4).

Property	Description	Type	Unit	Set	Get
stiffness	Stiffness of the effect’s surfaces	Float	[N/m]	V	V
dampfactor	Damping factor of the effect’s surfaces	Float	---	V	V
no_pull	Will the effect’s surfaces not pull the end effector while in contact?	Boolean	---	V	V
tang_damping	Tangential damping of the effect’s surfaces when in contact	Float	---	V	V

damping_forcemax	Maximum force applied by the surface while damping	Float	[N]	V	V
friction	Surface's friction	Float	---	V	V
ejection_velmax	Maximum velocity by ejection when inside the effect	Float	[m/s]	V	V
ejection_damping	Ejection damping when inside the effect	Float	[Ns/m]	V	V
outward_forcemax	Maximum force outwards	Float	[N]	V	V
powermax	Maximum power generated by the surface	Float	[W]	V	V

Table 4: *Description of the surface properties*

Example commands:
```
set myBlock no_pull true
“myBlock no_pull set”;
set mySphere damping_forcemax 3.0
“mySphere damping_forcemax set”;
get myFlatplane ejection_velmax
0.5;
```

3.5 Spring effect properties

The spring effect models a spring, acting on the HapticMaster End-Effector. Table 5 lists the properties of the spring effect. A spring effect can be set to act only in a certain direction, have a certain dead band (non-functional range), etc. If the direction is not set, then the spring is acting in every direction.

Property	**Description**	**Type**	**Unit**	**Set**	**Get**
stiffness	Stiffness of the spring	Float	[N/m]	V	V
dampfactor	Damping factor	Float	---	V	V
deadband	Non-functional length	Float	[m]	V	V
direction	Direction in which the spring is functional	Vector3D	1.0	V	V
maxforce	Maximum force the spring can apply	Float	[N]	V	V
dampglobal	Is the spring damped globally?	Boolean	---	V	V

Table 5: *Description of the spring properties*

Example commands:
```
create spring mySpring
"Effect spring with name mySpring created";
set mySpring dampfactor 0.7
“dampfactor set”;
get mySpring deadband
0.05;
```

```
get mySpring direction
[0.0,0.0,1.0];
get mySpring dampglobal
false;
```

3.6 Damper effect properties

The damper is an effect that acts as a damper on the movement of the HapticMaster. The effect of the damper can be defined in any direction of the haptic device's coordinate frame. Table 6: ***Description of the damper properties*** lists the properties of the damper effect.

Property	Description	Type	Unit	Set	Get
dampcoef	Damping coefficients	Vector3D	[Ns/m]	V	V

Table 6: *Description of the damper properties*

Example commands:

```
create damper myDamper
"Effect damper with name myDamper created";
set myDamper dampcoef [50.0,30.0,40.0]
“dampcoef set”;
get myDamper dampcoef
[30.0,20.0,30.0];
```

3.7 Bias Force effect properties

The bias force is an effect that allows the introduction of constant forces on the HapticMaster end-effector even when the end-effector is not actually touching anything within the virtual haptic environment. The bias force is actually a vector in 3d space. The force that is applied to the end-effector is in the direction of the set vector, and the total applied force equals to the length of the vector in [N]. **Error! Reference source not found.** lists the properties of the bias force effect.

Property	Description	Type	Unit	Set	Get
force	Bias force vector	Vector3D	[N]	V	V

Table 7: *Description of the bias force properties*

Example commands:

```
create biasforce myBiasForce
"Effect biasforce with name myBiasForce created";
set myBiasForce force [0.0,0.0,4.0]
“bias force set”;
get myBiasForce force
[0.0,2.0,0.0];
```

3.8 Shaker effect settings

The shaker is an effect that can be used for simple sine movements. It is realized by a spring that constantly moves its own position, so that the shaking criteria are met within the limitations (position / velocity / acceleration). That means that any of the three values can be sacrificed (not met) if one of the limitations is reached. The shaker also has a sweeping feature (isSweep), which allows you to transit between frequency1 and frequency2 while the shaker is shaking. This sweeping feature can be transiting again from frequency2 to frequency1 (isSweepUpAndDown) and can be repeated x times (sweepNumberOfRepeats).

Property	**Description**	**Type**	**Unit**	**Set**	**Get**
frequency1	Begin frequency	Float	[Hz]	V	V
frequency2	End frequency when sweeping	Float	[Hz]	V	V
direction	Oscillation direction vector (which axis will be influenced)	Vector3D	---	V	V
posmax	Maximum amplitude while oscillating	Float	[m]	V	V
velmax	Maximum velocity while oscillating	Float	[m/s]	V	V
accmax	Maximum acceleration while oscillating	Float	[m/s^2]	V	V
stiffness	Stabilizing spring stiffness	Float	[N/m]	V	V
dampfactor	Stabilizing spring damp factor	Float	---	V	V
deadband	Stabilizing spring deadband	Float	[m]	V	V
maxforce	Stabilizing spring maximum force	Float	[N]	V	V
endtime	End time of the shaker	Float	[s]	V	V
issweep	Should we sweep frequency1 → frequency2 ?	Boolean	---		V
issweepupanddown	Should we sweep frequenct1 → frequency2 → frequency1?	Boolean	---	V	V
sweepnumberofrepeats	How many times are we going to repeat the up-and-down	Integer	---	V	V

	sweep?				
wavesperhertz	# sine waves to go from freq1 to freq2	Float	---	V	V

Table 8: *Description of the shaker properties*

Example commands:

```
create shaker myShaker
"Effect shaker with name myShaker created";
set myShaker frequency1 5.0
“frequency1 set”;
get myShaker direction
[0.0,0.0,1.0];
get myShaker accmax
6.0;
```

3.9 Block effect settings

The block effect creates a box effect that can have different height, width and length. The effect’s surface properties can also be controlled by the surface properties that are shown at §3.4. Table 9 lists the properties of the block effect. All Surface settings apply to the block effect as well.

Property	**Description**	**Type**	**Unit**	**Set**	**Get**
size	Dimensions of the block	Vector3D	[m]	V	V
Surface properties	(See §3.4)	(See §3.4)		V	V

Table 9: *Description of the block properties*

```
create block myBlock
"Effect block with name myBlock created";
set myBlock stiffness 20000
“stiffness set”;
set myBlock no_pull true
“no_pull set”;
set myBlock size [0.2,0.1,0.1]
“size set”;
get myBlock size
[0.2,0.1,0.1];
```

3.10 Sphere effect settings

The sphere effect creates a spherical effect of a certain radius. The effect’s surface properties can also be controlled by the surface properties that are shown at §3.4. Table 10 lists the properties of the sphere effect.

Property	**Description**	**Type**	**Unit**	**Set**	**Get**
radius	Radius of the sphere	Float	[m]	V	V
Surface properties	(See §3.4)	(See §3.4)		V	V

Table 10: *Description of the sphere properties*

```
create sphere mySphere
```

```
"Effect sphere with name mySphere created";
set mySphere stiffness 20000
“sphere’s stiffness set”
set mySphere tang_damping 0.2
“tang_damping set”
set mySphere radius 0.1
“radius set”
get mySphere radius
0.1;
```

3.11 Flat Plane effect settings

The flat plane effect creates a plane, on which the point set by the position property of the effect exists. The orientation of the plane is defined by the normal vector of the plane. The behavior of the plane is set by the surface properties that are shown at §3.4. Table 11: ***Description of the flat plane properties*** lists the properties of the flat plane effect.

Property	Description	Type	Unit	Set	Get
normal	Normal perpendicular to the plane	Vector3D	---	V	V
Surface properties	(See §3.4)	(See §3.4)		V	V

Table 11: *Description of the flat plane properties*

```
create plane myPlane
"Effect flatplane with name myPlane created";
set myPlane stiffness 20000
"stiffness set";
set myPlane friction 0.2
"friction set";
set myPlane normal [0,0,1]
“flat plane’s normal set”;
get myPlane normal
[0,0,1];
```

3.12 Cylinder effect settings

The cylinder effect creates a cylindrical effect of a certain length and radius. By default the cylinder is centered around the Z-axis. The effect’s surface properties can also be controlled by the surface properties that are shown at §3.4. Table 12 lists the properties of the cylinder effect.

Property	Description	Type	Unit	Set	Get
radius	Radius of the cylinder	Float	[m]	V	V
length	Length of the cylinder	Float	[m]	V	V
Surface properties	(See §3.4)	(See §3.4)		V	V

Table 12: *Description of the cylinder properties*

```
create cylinder myCilinder
"Effect cylinder with name myCilinder created";
set myCylinder stiffness 20000
```

```
“stiffness set”;
set myCylinder powermax 10.0
“powermax set”;
set myCylinder radius 0.1
“radius set”;
get myCylinder length
0.2;
```

3.13 Capsule effect settings

The capsule effect creates a cylindrical effect which is rounded at its ends by the addition of hemispheres that have the same radius as the cylinder. By default the capsule is centered around the Z-axis. The effect's surface properties can also be controlled by the surface properties that are shown at §3.4. Table 13 lists the properties of the capsule effect.

Property	**Description**	**Type**	**Unit**	**Set**	**Get**
radius	Radius of the capsule	Float	[m]	V	V
length	Length of the cylindrical part	Float	[m]	V	V
Surface properties	(See §3.4)	(See §3.4)		V	V

Table 13: *Description of the capsule properties*

```
create capsule myCapsule
"Effect capsule with name myCapsule created";
set myCapsule stiffness 20000
“stiffness set”;
get myCapsule ejection_velmax
2.0;
set myCapsule radius 0.05
“radius set”;
get myCapsule length
0.1;
```

3.14 Torus effect settings

The torus effect creates a toroid effect of a certain length and radius. By default the torus center is around the Z-axis. The effect's surface properties can also be controlled by the surface properties that are shown at §3.4. Table 14 lists the properties of the torus effect.

Property	**Description**	**Type**	**Unit**	**Set**	**Get**
ring_radius	The ring (horizontal) radius of the torus	Float	[m]	V	V
tube_radius	The tube (vertical) radius of the torus	Float	[m]	V	V
Surface properties	(See §3.4)	(See §3.4)		V	V

Table 14: *Description of the torus properties*

```
create torus myTorus
"Effect torus with name myTorus created";
set myTorus stiffness 20000
“stiffness set”;
get myTorus ejection_velmax
2.0;
set myTorus ring_radius 0.1
“ring_radius set”;
get myTorus tube_radius
0.02;
```

4 HapticMASTER workspace

In Figure 1 and Figure 2 we show the workspace of the HapticMaster. For the HapticMaster we have defined a Cartesian coordinate system, where if you stand in front of the HapticMaster, the X-Axis is towards you, the Y-axis is to the right and the Z-axis is pointing upwards (See **Error! Reference source not found.**). Below is an example to get the position in this coordinate system:

```
get measpos
[0.00964991,0.0245587,-0.22857];
```

Internally we have a second coordinate system which is based on the Joints of the HapticMaster.
Here the **Z-axis** is similar as the **Z-axis** in the Cartesian coordinate system. It's positive upwards and Z = 0 in the middle of the height of the workspace.
Phi is the angle that the arm makes with respect to its centered position (along the positive X axis). Phi = 0 when the arm is in the middle. It's positive to the right of you (counter-clockwise from top)
R is the distance of the end effector from the vertical arched plane that divides the workspace in the middle of the radial depth (See **Error! Reference source not found.**). It's positive towards you.

X and Y are constantly calculated based on the measured R and Phi.

```
get measposjoint
[0.0102413,0.0481501,-0.22857];
```

We can also get the workspace in this joint coordinate system:

```
get workspace_r
[-0.188,0.5,0.188];
get workspace_phi
[-0.709,0,0.443];
get workspace_z
[-0.22,0,0.22];
```

The command "get workspace_r" returns a 3D vector.
The first element in the vector is the low-end stop (relative to the R zero point)
The second element is the distance from R zero point to the center of rotation of the arm
The third element is high-end stop (relative to the R zero point).
"get workspace_phi" and "get workspace_z" return the low-end stop, zero and the high-end stop.

Please notice the R and the Z axes are symmetric while the Phi axis is asymmetric**.**
In **Error! Reference source not found.** the workspace can be seen as X, Y and Z. In **Figure 2** the workspace can be seen as R, Phi and Z.

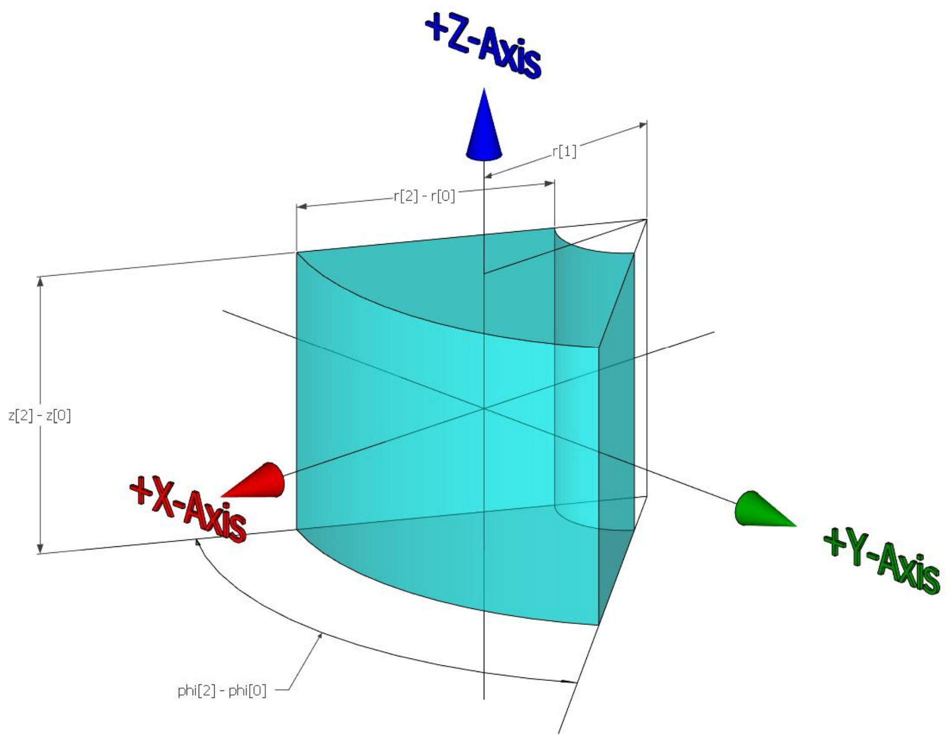

Figure 1 – The HapticMASTER's workspace seen as X, Y and Z axes (Cartesian)

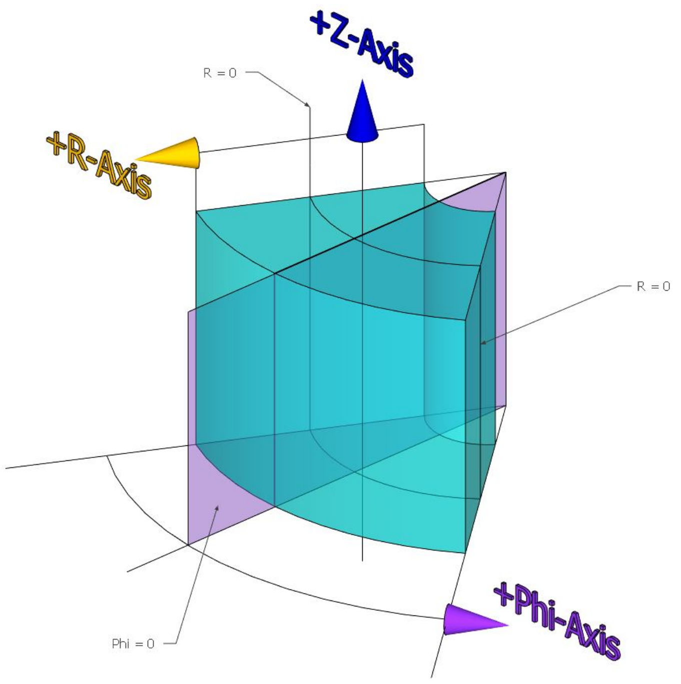

Figure 2 – The HapticMASTER's workspace seen as R, Phi and Z axes (Joints)

5 Effects with surfaces – Visually

In Figure 3 we show all available effects that have a surface. The figure also shows what the properties of the effects are.

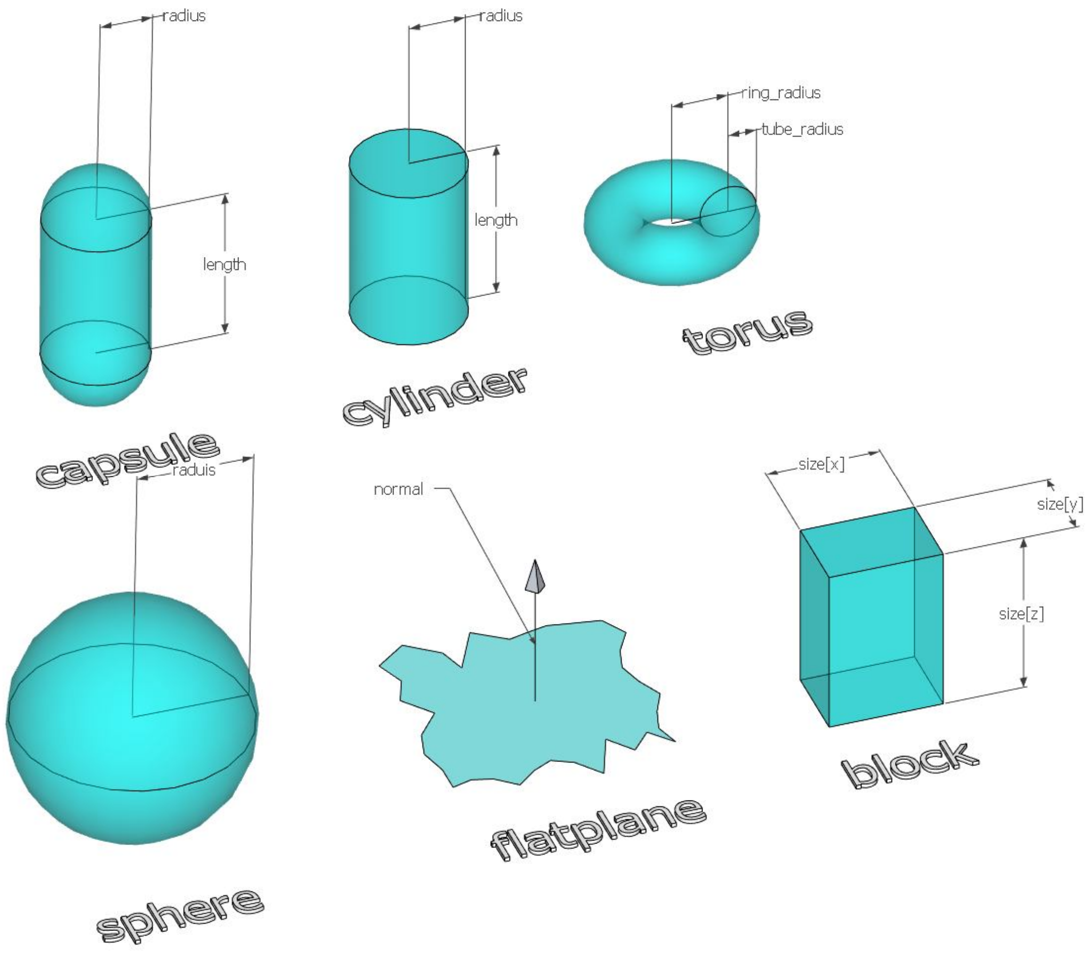

Figure 3 – HapticMASER's effects with surfaces

6 DataLogger

In the previous chapters we focused on the HapticAPI. Another importent interface to the HapticMaster is the data logger. The data logger is able to gather real-time data with time stamps on the HapticMaster and send this to the host computer. The data logger can log all variables used in the HapticAPI and the web interface. To use the web interface variables we can copy the address of that variable and add a '#' before the address. For example if we want to log the variable under Model, Cartesian, pos X, we use the following string as the variable that we want to add to the data logger:

```
#HapticMASTER.Model.Cartesian.pos X
```

Finally we have three extra parameters the data logger can log. If you want to add any of these to the data logger, simply use this name as the variable that you want to add to the data logger:

• `samplenr:`	Counter for the number of samples from the start of the log
• `sampletime:`	Seconds since the start of the log
• `currenttime:`	Seconds since the startup of the device

Finally you need to keep in mind that a data logger can log a maximum of 16 parameters at once. If you need more parameters you should start multiple data loggers. You can then synchronize the results by using the `currenttime` parameter.

6.1 DataLoggerRun.exe

The program DataLoggerRun.exe opens a data logger on the HapticMaster and writes the result to a specified file. The program's usage is as follows:

```
DataLoggerRun.exe ipadress [options] param_1 [param_2 [...[param_n]]]
Options:
-o   output file         [output.txt]
-s   subsample ratio     [1]
-d   duration in seconds [10.0]
-h   show this help
```

Where param_1 to param_n are the parameters that should be logged. An example:

```
DataLoggerRun.exe 10.30.211.1 –o datalog.txt –s 2 -d 60.0 measpos measvel
```

This call will start a data logger on the HapticMaster with ipadress 10.30.211.1. It will write to the file datalog.txt, logs once in two samples, for one minute. The logged data will be three values for the measured position and three values for the measured velocity.

The resulting file of this will be something like:

```
% [HEADER]
% Filename  : output.txt
%
% [DATA]
0.009650 0.024559 -0.228575 -0.000000 0.000000 0.000000
…
```

So first are x, y, and z values of the position followed by x, y and z of the velocity.

6.2 haDataLogger Functions

The data logger can also be interfaced though functions in the HapticAPI. The functions will be explained in the sections below.

6.2.1 haDataLoggerOpen

The haDataLoggerOpen function allows a host computer to create a data logger on the HapticMaster. The syntax of the function is the following:

```
dataLogger = haDataLoggerOpen ( inAddress )
```

Inputs:

`inAddress`	A **char array** with an IP4 address of the haptic device, which also can be predefined.

Outputs:

`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer. On failure to connect the function will return -1.

6.2.2 haDataLoggerClose

The haDataLoggerClose function allows a host computer to disconnect from a data logger on the HapticMaster. The syntax of the function is the following:

```
returnValue = haDataLoggerClose ( dataLogger )
```

Inputs:

`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.

Outputs:

`returnValue`	A variable of type **int**. On failure to close the data logger the function will return -1.

6.2.3 haDataLoggerConfigure

The haDataLoggerConfigure function allows the user to configure the data logger flush sizes and sampling ratio. The syntax of the function is as follows:

```
returnValue = haDataLoggerConfigure (    dataLogger,
                                        inFlushSizeMax,
                                         inSubSampleRatio )
```

Inputs:

`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.
`inFlushSizeMax`	**Integer** value that determines the maximum number of samples that will be flushed by calling haDataLoggerFlush. If the buffer has fewer samples, only these samples will be flushed. If it has more samples, only *inFlushSizeMax* samples will be flushed.
`inSubSampleRatio`	**Integer** value that determines how often will be sampled. Value is 1? each cycle generates a sample, Value is 2? One cycle generates a sample and one cycle not

Outputs:

`returnValue`	A variable of type int. On failure to configure the data logger the function will return -1.

6.2.4 haDataLoggerAllocMatrix

The haDataLoggerAllocMatrix function allocates memory where we can flush the data logger to. The syntax of the function is as follows:

```
haDataLoggerAllocMatrix (     inFlushSizeMax,
                              inColumnCount,
                              ioMatrix )
```

Inputs:

`inFlushSizeMax`	**Integer** value that determines the number of samples (lines) that will be allocated in the host computer.
`inColumnCount`	**Integer** value that contains the number of variables (columns) that we log.
`ioMatrix`	**Reference to a float pointer pointer** where data will be allocated to flush the samples to.

6.2.5 haDataLoggerFreeMatrix

The haDataLoggerFreeMatrix function releases the memory allocated for the flushed samples. The syntax of the function is as follows:

```
haDataLoggerFreeMatrix ( inColumnCount,
                         ioMatrix )
```

Inputs:

`inColumnCount`	**Integer** value that contains the number of columns we log.
`ioMatrix`	**Reference to a float pointer pointer** that was allocated by haDataLoggerAllocMatrix.

6.2.6 haDataLoggerAddParameter

The haDataLoggerAddParameter function adds a parameter to the data logger. The syntax of the function is as follows:

```
returnValue = haDataLoggerAddParameter ( dataLogger
                                         inParameter
                                         outColumnCount )
```

Inputs:

`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.
`inParameter`	**Char array** determining the property as described previously in §6

Outputs:

`returnValue`	**Integer** value that is -1 if adding the parameter failed
`outColumnCount`	**Integer** value that contains the total number of columns that we will be logging after adding this parameter.

6.2.7 haDataLoggerRemoveParameter

The haDataLoggerRemoveParameter function removes a parameter from the data logger. The syntax of the function is as follows:

```
returnValue = haDataLoggerRemoveParameter (   dataLogger,
                                               inParameter,
                                               outColumnCount )
```

Inputs:

`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.
`inParameter`	**Char array** determining the property as described previously.

Outputs:

`returnValue`	**Integer** value that is -1 if removing the parameter failed
`outColumnCount`	**Integer** value that contains the total number of columns that we will be logging after removing this parameter.

6.2.8 haDataLoggerRemoveAllParameters

The haDataLoggerRemoveAllParameters function removes all parameters from the DataLogger. The syntax of the function is as follows:

```
returnValue = haDataLoggerRemoveAllParameters ( dataLogger )
```

Inputs:

`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.

Outputs:

`returnValue`	**Integer** value that is -1 if removing all parameters failed

6.2.9 haDataLoggerStart

The haDataLoggerStart function makes the data logger start logging. The syntax of the function is as follows:

```
returnValue = haDataLoggerStart ( dataLogger );
```

Inputs:

	`dataLogger`	A variable of type **long**, unique for each dataLogger that is connected to the host computer.

Outputs:

	`returnValue`	**Integer** value that is -1 if starting the datalogger failed

6.2.10 haDataLoggerStop

The haDataLoggerStop function makes the data logger stop logging. The syntax of the function is as follows:

```
returnValue = haDataLoggerStop ( dataLogger )
```

Inputs:

	`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.

Outputs:

	`returnValue`	**Integer** value that is -1 if stopping the data logger failed

6.2.11 haDataLoggerGetStatus

The haDataLoggerGetStatus function gives the status of the data logger. The syntax of the function is as follows:

```
returnValue = haDataLoggerGetStatus (    dataLogger,
                                         outLogging,
                                         outBufferSize,
                                         outFlushSizeMax,
                                         outColumnCount );
```

Inputs:

`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.

Outputs:

`returnValue`	**Integer** value that is -1 if getting the status failed
`outLogging`	**Pointer to a Boolean** which will show whether the data logger is logging at the moment.
`outBufferSize`	**Pointer to an integer** which will give the number of samples present at the moment in the data logger buffer.
`outFlushSizeMax`	**Pointer to an integer** which will give the maximum flush size (configured).
`outColumnCount`	**Pointer to an integer** which will give the number of variables (columns) that are being logged.

6.2.12 haDataLoggerFlushMatrix (For C++ function calls)

The haDataLoggerFlushMatrix function is used to flush the data of the data logger to the host computer in a matrix form (2D array). The syntax of the function is as follows:

```
returnValue = haDataLoggerFlushMatrix (  dataLogger,
                                         outMatrix );
```

Inputs:

`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.

Outputs:

`returnValue`	**Integer** value that is -1 if flushing failed
`outMatrix`	**Reference** to the memory allocated by haDataLoggerAllocMatrix where the data will be flushed to.

6.2.13 haDataLoggerFlushVector (For Matlab function calls)

The haDataLoggerFlushVector function is used to flush the data of the data logger to the host computer in a vector form (1D array). The syntax of the function is as follows:

```
returnValue = haDataLoggerFlushVector (  dataLogger,
                                         outVector );
```

Inputs:

`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.

Outputs:

`returnValue`	**Integer** value that is -1 if flushing failed
`outVector`	**Float pointer** to allocated memory where the data will be flushed to.

6.2.14 haDataLoggerStartThread

The haDataLoggerStartThread function makes the data logger start logging and flushing the results automatically to a file. The syntax of the function is as follows:

```
returnValue = haDataLoggerStartThread (  dataLogger,
                                         inFileName,
                                         outProcessHandle );
```

Inputs:

`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.
`inFileName`	**Char array** containing the name of the file where the results will be written.

Outputs:

`returnValue`	**Integer** value that is -1 if starting the data logger failed
`outProcessHandle`	A variable of type **long**, unique to the process that handles the data logger.

6.2.15 haDataLoggerStopThread

The haDataLoggerStop function makes the data loggerthread stop logging and kills the automatic flushing process. The syntax of the function is as follows:

```
returnValue = haDataLoggerStopThread (   dataLogger,
                                         inProcessHandle);
```

Inputs:

	`dataLogger`	A variable of type **long**, unique for each data logger that is connected to the host computer.
	`inProcessHandle`	A variable of type **long**, unique to the process that handles the data logger.

Outputs:

	`returnValue`	**Integer** value that is -1 if stopping the data logger failed

Document Revision History

Version Nr	**Issue Date**	**Comments**
V0.0	19 December 2001	First draft completed
V2.0	29 November 2010	Updated whole document to fit String commands API
V2.1	02 January 2012	Updates with respect to quaternions, new objects
V2.2	16 April 2012	Updated reference manual to fit RT 4.1.2-11
V2.3	18 November 2013	Updated reference manual to fit RT 4.2. Added Telnet, Matlab, DataLogger.