



HAPTION
Atelier Relais
ZA – Route de Laval
53210 SOULGE SUR OUETTE
FRANCE
<http://www.haption.com>
contact@haption.com

VIRTUOSE API V3.60

DOCUMENTATION



	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 2

TABLE OF CONTENTS

1	INTRODUCTION.....	3
1.1	WHAT IS THE VIRTUOSE API?	3
1.2	COMPOSITION OF THIS DOCUMENTATION	3
2	INSTALLATION MANUAL	4
3	USER MANUAL	5
3.1	CONTENTS OF THE VIRTUOSE API.....	5
3.2	CONVENTIONS	5
3.2.1	Programming	5
3.2.2	Interaction mode.....	5
3.2.3	Definition of reference frames.....	5
3.2.4	Physical units	6
3.2.5	Expression of geometric entities.....	7
3.3	IMPLEMENTATION OF THE VIRTUOSE API	7
3.3.1	Initialisation	7
3.3.2	Use as pointer.....	9
3.3.3	Object attachment.....	10
3.3.4	Timing	11
3.3.5	Evolutions of v2.28.....	12
3.4	PROBLEM SOLVING.....	13
3.4.1	Compatibility with versions 1.x.....	13
3.4.2	Obsolete functions.....	13
3.4.3	Instability.....	13
3.4.4	Abnormal sticking.....	13
3.4.5	Low stiffness	14
3.4.6	Low force feedback.....	14
4	CHANGE OF CONVENTIONS.....	15
4.1	EXAMPLE.....	15
5	REFERENCE MANUAL	17
5.1	FUNCTION INDEX	17
6	GLOSSARY.....	92

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 3

1 INTRODUCTION

1.1 What is the VIRTUOSE API?

The VIRTUOSE API (Application Programming Interface) is a library of functions written in the C programming language, allowing programmers to interface their applications with haptic devices of the VIRTUOSE series.

The VIRTUOSE API supports the following devices:

- VIRTUOSE 6D35-45
- VIRTUOSE 3D35-40
- VIRTUOSE 3D10-20
- VIRTUOSE DESKTOP
- VIRTUOSE INCA

It is able to control several devices at the same time, as well as the simulation software itself.


It is available for different hardware architectures and operating systems:

- PC-type computer running Microsoft Windows 98/NT/2000/XP/7
- PC-type computer running Linux (kernel 2.4 and 2.6)

1.2 Composition of this documentation

This document includes the following chapters:

1. Introduction (this chapter)
2. Installation manual
3. User manual
4. Reference manual
5. Glossary


	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 4

2 INSTALLATION MANUAL

The VIRTUOSE API is a library of C functions. Its installation consists simply to including files into the development environment. Those header et binary files depend on the operating system:

- For Microsoft Windows 98/NT/2000/XP/7 :
 - virtuoseAPI.h
 - virtuoseDLL.lib
 - virtuoseAPI.dll

- For Linux:
 - virtuoseAPI.h
 - virtuoseAPI.so
 - libvirtuose.a

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 5

3 USER MANUAL

3.1 Contents of the Virtuose API

The VIRTUOSE API is composed of the following files:

- One header file for C/C++ to be included in your source code: VirtuoseAPI.h
- One static library: VirtuoseDLL.lib (Windows) or libvirtuose.a (Linux)
- One dynamic library: VirtuoseAPI.dll (Windows) or virtuoseAPI.so (Linux)

3.2 Conventions

3.2.1 Programming

The functions of VIRTUOSE API respect the following rules:

- The names of functions are in English language; they start with the letters “*virt*” and don't include any underscore, but all following words are capitalized (ex : “*virtGetPosition*”).
- Apart from the two functions “*virtOpen*” and “*virtGetErrorMessage*”, all functions need as first parameter an object of type “*VirtContext*”, which is created by calling “*virtOpen*” and destroyed with “*virtClose*”.
- Apart from “*virtOpen*”, “*virtGetErrorCode*” and “*virtGetErrorMessage*”, all functions return 0 in case of success and -1 otherwise.
- The functions use C prototypes; the header file VirtuoseAPI.h includes an *extern "C"* pragma for use with C++ applications.

3.2.2 Interaction mode


The VIRTUOSE library implements different control modes for the haptic device. The choice of the control mode depends on the application:

1. Force/position control: the application sends forces and torques to the device and reads the position and speed of the end-effector frame.
2. Position/force control: this advanced control mode allows direct coupling with virtual objects; in that case, the application sends the position and speed of the center of the object to the device, and reads the forces and torques to be applied to the object for dynamic integration. Stiffness and damping are calculated by the embedded software, knowing the mass and inertia of the object, in order to ensure control stability.
3. Position/force with virtual guides: this is the same as above, with addition of virtual guides (e.g. fixed translation, fixed rotation, etc.).

3.2.3 Definition of reference frames

All geometric entities (position, speed, forces and torques, etc.) are defined with respect to a reference frame.

By definition, all reference frames used by the VIRTUOSE API are right-handed. This is different to the conventions used in several graphic libraries. Chapter 4 deals with the change of conventions. By default, all reference frames are defined as follows:

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 6

- The origin is the intersection point of axis 1 (base rotation) and axis 2 (first vertical movement) of the haptic device.
- The X axis is horizontal, aligned with the central position of axis 1 and of the haptic device, pointing towards the user.
- The Y axis is horizontal, orthogonal to the X axis and pointing to the right of the user.
- The Z axis is vertical, pointing upwards.

The library defines the following reference frames:

1. The environment frame corresponds to the origin of the virtual scene; it is specified by the software application independently of the Virtuose API.
2. The observation frame corresponds generally to the position of the camera, and is defined with respect to environment frame.
3. The base frame represents the center of the haptic device, and is defined with respect to the observation frame.
4. The tool frame corresponds to the base of the tool fixed at the end of the haptic device, and is defined with respect to the environment frame.
5. The end-effector frame corresponds to the position of the user hand on the device, taking into account the geometry of the tool, and is defined with respect to tool frame.

The position of the following frames can be defined only once using the API:

- Base frame (with respect to the observation frame)
- End-effector frame (with respect to the tool frame)

The position of the following frames can be modified dynamically using the API:

- Observation frame (with respect to the environment frame)

The position of the following frame cannot be modified:

- Tool frame (with respect to the environment frame)

3.2.4 Physical units

All values used in the API are expressed in physical units using metric conventions:

- Durations in seconds (s)
- Dimensions in meters (m)
- Angles in radians (rad)
- Linear velocities in meters per second (m.s^{-1})
- Angular velocities in radians per second (rad.s^{-1})
- Forces in Newtons (N)
- Torques in Newton-meters (N.m)
- Masses in kilogrammes (kg)
- Inertia components in kg.m^2

3.2.5 Expression of geometric entities

The API uses different geometric entities:

- Positions are expressed as displacement vectors with seven components, i.e. one translation term (x, y, z) followed by one rotation term in the form of a normalized quaternion (qx, qy, qz, qw) (cf. glossary).
- Velocities are expressed as cinematic tensors (vx, vy, vz, wx, wy, wz) (cf. glossary)
- Efforts are expressed as dynamic tensors (fx, fy, fz, cx, cy, cz) (cf. glossary)

3.3 Implementation of the VIRTUOSE API

3.3.1 Initialisation

The first step in use of VIRTUOSE API is to establish the communication with the haptic device. The `virtOpen` function carries out that operation, allocates the memory for storing internal data and returns a pointer of type `VirtContext`. This pointer must be used for calling all other functions: In case several haptic devices are connected to the application, the `VirtContext` object also identifies which VIRTUOSE to address a request to.

It is essential to test the return value of the `virtOpen` function, the API works well only if the opening of the communication is successful:

```
VirtContext VC;
VC = virtOpen("192.168.1.1");
if (VC == NULL)
{
    fprintf(stderr, "Erreur dans virtOpen: %s\n",
            virtGetErrorMessage(virtGetErrorCode(NULL));
    return -1;
}
```

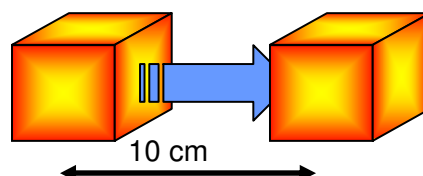
Afterwards, we advise to do explicit configuration of the control mode, as the default parameters possibly don't correspond to your application needs:

1. The speed factor (`virtSetSpeedFactor` function)

The role of speed factor is to modify the amplitude of translations during the simulation. This function is useful when the virtual scene is much larger or smaller as the workspace of the Virtuose haptic device.

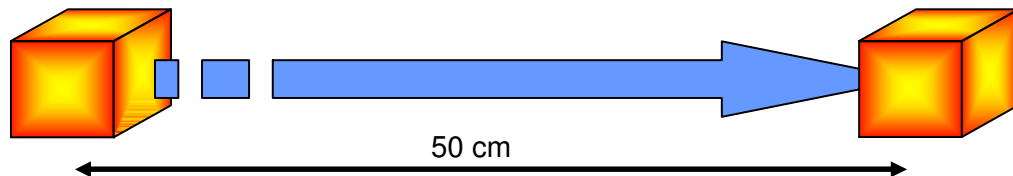
The following figure illustrates the effect of the speed factor on a virtual object:

Case 1: Speed factor of 1.0



For a 10 cm horizontal displacement of haptic interface, the displacement of the virtual box is the same.

Case 2: Speed factor of 5.0



For a 10 cm horizontal displacement of haptic interface, the displacement of the virtual box is 5 times more, i.e. 50 cm.

2. The force factor (`virtSetForceFactor` function)

The role of the force factor is to modify the level of forces to be exerted by the user when moving virtual objects, especially in the case of large or heavy objects. A force factor larger than 1 will amplify the forces computed by the simulation, while a value lower than 1 will amplify the forces exerted by the user on the virtual environment.

3. The indexing mode (`virtSetIndexingMode` function)

When the user reaches the limits of the device workspace, he can use the indexing function (also called offset) in order to come back to a more appropriate position without moving the virtual object he is attached to. The indexing function is also activated when the user lets go of the proximity sensor (also called dead-man sensor), or when the power is switched off.

Three different modes of indexing are defined:

- ✎ `INDEXING_ALL` authorizes indexing on all movements, i.e. rotations and translations.
- ✎ `INDEXING_TRANS` authorizes indexing only on translations, i.e. the orientation of the object is always identical to that of the device end-effector
- ✎ `INDEXING_NONE` forbids indexing on all movements.

4. The simulator integration timestep (`virtSetTimeStep` function)

5. The position of the base reference frame with respect to the observation reference frame (`virtSetBaseFrame` function)

6. The position of the observation reference frame with respect to the environment reference frame (`virtSetObservationFrame` function)

7. The type of control mode (`virtSetCommandType` function)

The following source code represents very common settings:

```
float identity[7] = {0.0f,0.0f,0.0f,0.0f,0.0f,0.0f,1.0f};
virtSetIndexingMode(VC, INDEXING_ALL);
virtSetForceFactor(VC, 1.0f);
virtSetSpeedFactor(VC, 1.0f);
```



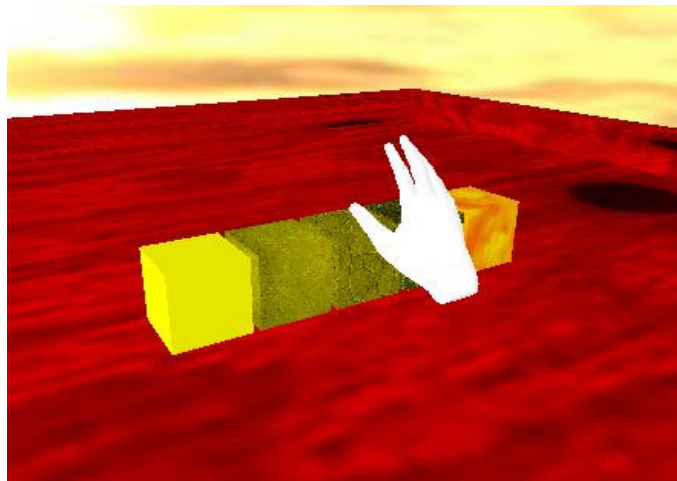
```
virtSetTimeStep(VC, 0.003f);
virtSetBaseFrame(VC, identity);
virtSetObservationFrame(VC, identity);
virtSetCommandType(VC, COMMAND_TYPE_VIRTMECH);
```

Finally, the initialisation phase ends with the enabling of force-feedback (`virtSetPowerOn` function). After that step, the user can activate the force-feedback at any time, by switching the power switch to ON and engaging the proximity sensor:

```
virtSetPowerOn(VC, 1);
```

3.3.2 Use as pointer

In many applications, it is necessary to "pointer mode", in which the current position of the haptic device is shown on the screen as a graphic object (also called "avatar"). This is useful for navigating in the virtual scene in order to choose an object to grasp, or else to call a function using a contextual menu.



We define a simple update function, which will be called at regular intervals, in order to read the position of the haptic device and send new control values. The function will be called by the simulation, for example as a callback.

In simple `COMMAND_TYPE_IMPEDANCE` control mode, the function must set the control force to 0, and return the position of the end-effector. It can be written as follows:

```
int update_avatar(VirtContext VC, float *position)
{
    float null [6] = {0.0f,0.0f,0.0f,0.0f,0.0f,0.0f};
    virtSetForce(VC, null);
    virtGetPosition(VC, position);
    return 0;
}
```

In `COMMAND_TYPE_VIRTMECH` control mode, the function must read the position and speed of the end-effector, and send it back to the device (as a consequence, the position and speed errors are both null, and no force is generated):

```
int update_avatar(VirtContext VC, float *position)
{
    float speed[6];
    virtGetPosition(VC, position);
    virtGetSpeed(VC, speed);
    virtSetPosition(VC, position);
    virtSetSpeed(VC, speed);
    return 0;
}
```

In both cases, the function returns the current position of the avatar in the parameter `position`, in translation and rotation. Its value is affected by the choice of the base and observation reference frames, and by the indexing: when the offset push-button is pressed, the avatar does not follow the movements of the end-effector anymore.

3.3.3 Object attachment

In `COMMAND_TYPE_VIRTMECH` control mode, it is possible to attach the end-effector of the VIRTUOSE directly to a virtual object, and the haptic device is then seen by the simulation as a dynamic bilateral constraint applied on the object.

The attachment to a virtual object is done as follows:

1. The function `virtAttachVO` carries out the attachment, taking into account the mass and inertia of the object as well as the integration timestep of the simulation, in order to compute optimal values for the control parameters, guaranteeing the stability.
2. The position and speed of the reference point of the object are then sent as control values using the functions `virtSetPosition` and `virtSetSpeed`.

The following source code implements the attachment of an object of mass m , of inertia $mxmymz$, whose center of mass is at the position P moving with speed V :

```
virtAttachVO(VC, m, mxmymz);
virtSetPosition(VC, P);
virtSetSpeed(VC, S);
```

In order to enable the motion of the virtual object, we define an update function to be called at a fixed rate by the simulation, typically as a callback:

```
int update_object(VC, float *P, float *S, float *force)
{
    virtSetPosition(VC, P);
    virtSetSpeed(VC, S);
    virtGetForce(VC, force);
    return 0;
}
```

After calling the function, the simulation should add the value of `force` to the constraints active on the object, before integrating the laws of physics.


3.3.4 Timing

Sometimes it is necessary to add software temporizations between writing and reading operations of a parameter, as the embedded software controller of the Virtuose needs time to taking the new value into account. This concerns:

- the base frame of a virtual mechanism,
- the sequences of two virtual mechanisms

Example: Cartesian movement from current position of end-effector 20 cm along X axis, then a rotation of X axis from effector.

```
float virtVmBaseFrame[7];
virtVmSetType(VC, VM_TYPE_CartMotion);
// the current position of end-effector becomes
// the base frame of the virtual mechanism
virtVmSetBaseFrameToCurrentFrame(VC);
Sleep(10);
// new modification of base frame
virtVmGetBaseFrame(VC, virtVmBaseFrame);
virtVmBaseFrame[0] += 0.2;
virtVmSetBaseFrame(VC, virtVmBaseFrame);
// activation of guide
virtVmActivate(VC);
// wait bound
virtVmWaitUpperBound(VC);
virtVmDeactivate(VC);
Sleep(10);
// new mechanism, rotation of x axis
virtVmSetType(VC, VM_TYPE_Rx);
virtVmActivate(VC);
```

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 12

3.3.5 Evolutions of v3.60

1. A new virtual guide is implemented; a plane mechanism allows the user to define a plane with the Ox and Oy axis of base frame. The user is forced on the plane after activation of the mechanism.
2. A second virtual guide, the crank mechanism. The arm of lever is fixed at activation time at the position of the end-effector with respect to Ox axis of virtual mechanism base frame.
3. During the simulation, the speed factor can be modified and the user can work with different scale ratios. This is relevant in case of a zoom on the screen camera.
4. The observation frame can be modified (rotation only) during the simulation. When attached to an object, the user can feel inertial forces if the observation frame turns quickly.
5. A virtual object can be attached at any position, different from its center of mass. For example, the handle for a hammer. As a result, the user feels a lever in the forces.
6. Additionnally to the position/force control mode, a new control mode has been introduced, which is called hybrid position/speed control. Inside a sphere placed at the center of the workspace, the control mode is identical to position/force. However, outside the sphere, the position moves proportionally to the distance to the sphere surface, in the same direction. He result is an amplification of the device workspace, so that it is not necessary to use the indexing function anymore.
7. There are an new indexing mode, INDEXING_ALL_FORCE_FEEDBACK_INHIBITION. It allows to inhibit force feedback while the user presses the offset button. In the case of a virtual object in collision, the force disappears and comes back gradually.
8. The connexion with the device haptic is monitored only while the haptic loop is running. The application can connect to a second device after initialisation of the first haptic device and before starting the haptic loop.
9. It is possible to inhibit the indexing function, in order to work with a co-location of the virtual object and the end-effector of the Virtuose. As it is still possible to move the device by switching off force-feedback, an automatic movement is carried out when the force-feedback is activated again, bringing the end-effector back to the position of the virtual object.
10. A new control mode allows to move the mouse pointer with the Virtuose haptic device. The user enables and disables it with a double clic on one of the push-buttons. In this mode, the end-effector is forced on a vertical plane corresponding to the screen, the virtual objet possibly attached doesn't move. Two push-buttons on the Virtuose simulate the right and the left buttons of the mouse. The mouse connected on the PC is always available.

3.4 Problem solving

We describe below some typical problems, which arise when using the API. For each of them, we give solutions and rules to follow.

3.4.1 Compatibility with versions 1.x

The main modifications in the VIRTUOSE API are the following:

1. Control modes `COMMAND_TYPE_FORCE` and `COMMAND_TYPE_POSITION` have been replaced with `COMMAND_TYPE_IMPEDANCE` and `COMMAND_TYPE_VIRTMECH`.
2. The state of the push-buttons is accessed by one function only (compared to one function per push-button in version 1.x)
3. Some parameters, especially the position of the base reference frame, can be modified only before selecting the control mode.

3.4.2 Obsolete functions

The functions `virtSetLimitTorque` and `virtGetLimitTorque` are obsolete, and should not be used anymore. The forces can be reduced by calling the function `virtSaturateTorque`, which has been significantly improved.

3.4.3 Instability

The VIRTUOSE API computes optimal values for the control parameters, with respect to two criteria:

1. Global stability of the system (simulation – virtual object – haptic device)
2. Stiffness of the haptic device control loop

Should an instability occur nonetheless, it is advisable to verify several constraints:

- Is the integration step of the simulation exactly identical to that given to the API by calling the function `virtSetTimeStep`?
- Is the simulation on time, i.e. is there no delay between the simulated time and the real time?
- Are the mass and inertia parameters of the virtual object exactly identical to those given to the API?

Moreover, some collision detection algorithms give very noisy results, with ill-defined surface normals, which can be misinterpreted as instability.


3.4.4 Abnormal sticking

This phenomenon is characteristic of a loss of efficiency in the connection with the Virtuose. This is particularly true when the Virtuose is situated on a network branch shared with many other workstations, or when the Virtuose and the simulator are not on the same branch but talk to each other over several Ethernet relays.

In all those cases, it is necessary to modify the network organization, the best solution being to dedicate only network interface to the Virtuose.

3.4.5 Low stiffness

This phenomenon appears when the Virtuose is attached to very large objects, or else when the movement scale factor (`virtSetSpeedFactor`) is inadequate. As a consequence, the

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 14

movement of the attached object is difficult to control, because of its very high mass and inertia compared to the forces exerted by the VIRTUOSE.

If that is the case, then it is advisable to reduce the mass and inertia of the attached object, either by modifying its size (global scale factor for the whole scene), or by diminishing its density. It is also possible to reduce the force scale factor (`virtSetForceFactor`), which results in amplifying the forces exerted by the VIRTUOSE, (see below).

3.4.6 Low force feedback

This phenomenon appears when the force scale factor is too low (`virtSetForceFactor`). It can also occur with some combinations of mass, inertia and movement scale factor.

In all cases, it is advisable to raise the force factor, or else to modify the object in order to bring it closer to the optimal values (sphere of 10 cm and 300 g, speed factor of 1).

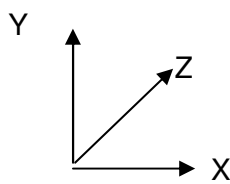
4 CHANGE OF CONVENTIONS

A classical problem is the transformation between conventions used in computer graphics (left-handed coordinate system, Z pointing towards the back of the screen) and those used in robotics (right-handed coordinate system, Z vertical and pointing upwards).

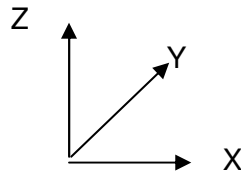
We propose here below a simple conversion scheme, illustrated on the Vortex™ SDK of the company CMLabs Simulations Inc.

4.1 Example

We define the following conversion functions, which switch axes Y and Z, in order to transform between computer graphics robotics conventions to and fro:



Vortex coordinates



Virtuose coordinates

- Vortex positions are defined as positions and quaternions.
- Vortex speeds are defined as linear speed and angular speed.

```
void transformVortexPosQuatToVirtPosEnv(float *vortexPos,
float *vortexQuat,
float *virtPosEnv)
{
    ..... virtPosEnv[0] = vortexPos[0];
    ..... virtPosEnv[1] = vortexPos[2];
    ..... virtPosEnv[2] = vortexPos[1];
    ..... virtPosEnv[3] = -vortexQuat[1];
    ..... virtPosEnv[4] = -vortexQuat[3];
    ..... virtPosEnv[5] = -vortexQuat[2];
    ..... virtPosEnv[6] = vortexQuat[0];
}

void transformVirtPosEnvToVortexPosQuat(
    ..... float *virtPosEnv,
    ..... float *vortexPos,
    ..... float *vortexQuat )
{
    ..... vortexPos[0] = virtPosEnv[0];
    ..... vortexPos[1] = virtPosEnv[2];
    ..... vortexPos[2] = virtPosEnv[1];
    ..... vortexQuat[0] = virtPosEnv[6];
    ..... vortexQuat[1] = -virtPosEnv[3];
    ..... vortexQuat[2] = -virtPosEnv[5];
    ..... vortexQuat[3] = -virtPosEnv[4];
}
```



```
void transformVortexLinAngVelToVirtSpeedEnv(float *vortexLinVel,
..... float *vortexAngVel,
..... float *virtSpeedEnv)
{
.... virtSpeedEnv[0] = vortexLinVel[0];
.... virtSpeedEnv[1] = vortexLinVel[2];
.... virtSpeedEnv[2] = vortexLinVel[1];
.... virtSpeedEnv[3] = -vortexAngVel[0];

.... virtSpeedEnv[4] = -vortexAngVel[2];
.... virtSpeedEnv[5] = -vortexAngVel[1];
}

void transformVirtTorsToVortexForceTorque(float *virtTors,
..... float *vortexForce,
..... float *vortexTorque)
{
..... vortexForce[0] = virtTors[0];
..... vortexForce[1] = virtTors[2];
..... vortexForce[2] = virtTors[1];
..... vortexTorque[0] = -virtTors[3];
..... vortexTorque[1] = -virtTors[5];
..... vortexTorque[2] = -virtTors[4];
}


void transformVortexInertiaTensorToVirtInertiaTensor(float *vortexTensor,
..... float *virtTensor)
{
..... virtTensor[0] = vortexTensor[0];
..... virtTensor[1] = vortexTensor[2];
..... virtTensor[2] = vortexTensor[1];
..... virtTensor[3] = vortexTensor[6];
..... virtTensor[4] = vortexTensor[8];
..... virtTensor[5] = vortexTensor[7];
..... virtTensor[6] = vortexTensor[3];
..... virtTensor[7] = vortexTensor[5];
..... virtTensor[8] = vortexTensor[4];
}
```


5 REFERENCE MANUAL

5.1 Function index

virtActiveRotationSpeedControl	103
virtActiveSpeedControl	82
virtAddForce	92
virtAttachVO	19
virtClose	20
virtConvertDeplToHomogeneMatrix	88
virtConvertHomogeneMatrixToDepl	89
virtConvertRGBToGrayscale	72
virtDeactiveRotationSpeedControl	104
virtDeactiveSpeedControl	83
virtDetachVO	21
virtDisableControlConnexion	76
virtDisplayHardwareStatus	22
virtGetAlarm	81
virtGetArticularPosition	98
virtGetArticularPositionOfAdditionalAxe	93
virtGetArticularSpeed	100
virtGetArticularSpeedOfAdditionalAxe	95
virtGetAvatarPosition	67
virtGetBaseFrame	23
virtGetButton	24
virtGetCatchFrame	85
virtGetCommandType	25
virtGetControlerVersion	105
virtGetDeadMan	26
virtGetEmergencyStop	27
virtGetError	28
virtGetErrorCode	29
virtGetErrorMessage	31
virtGetForce	34
virtGetForceFactor	35
virtGetIndexingMode	36
virtGetMouseState	87
virtGetObservationFrame	37
virtGetPhysicalPosition	66
virtGetPosition	38
virtGetPowerOn	39
virtGetSpeed	40
virtGetSpeedFactor	41
virtGetTimeLastUpdate	42
virtGetTimeoutValue	43
virtGetTimeStep	33
virtGetTrackball	90
virtGetTrackballButton	91
virtIsInBounds	80

virtIsInShiftPosition	86
virtIsInSpeedControl	106
virtOpen	44
virtSetArticularForce	102
virtSetArticularForceOfAdditionalAxe	97
virtSetArticularPosition	99
virtSetArticularPositionOfAdditionalAxe	94
virtSetArticularSpeed	101
virtSetArticularSpeedOfAdditionalAxe	96
virtSetCatchFrame	84
virtSetCommandType	46
virtSetDebugFlags	47
virtSetForce	48
virtSetForceFactor	49
virtSetForceInSpeedControl	107
virtSetIndexingMode	50
virtSetObservationFrame	51
virtSetObservationFrameSpeed	52
virtSetPeriodicFunction	77
virtSetPosition	53
virtSetSpeed	54
virtSetSpeedFactor	55
virtSetTexture	73
virtSetTextureForce	74
virtSetTimeoutValue	56
virtSetTimeStep	32
virtSetTorqueInSpeedControl	108
virtStartLoop	78
virtStopLoop	79
virtTrajRecordStart	62
virtTrajRecordStop	63
virtVmActivate	59
virtVmDeactivate	60
virtVmGetBaseFrame	71
virtVmGetTrajSamples	65
virtVmSetBaseFrame	70
virtVmSetDefaultToCartesianPosition	68
virtVmSetDefaultToTransparentMode	69
virtVmSetSamplingTimeStep	61
virtVmSetType	57
virtVmStartTrajSampling	64
virtVmWaitUpperBound	75

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 19

NAME

virtAttachVO – attaches the VIRTUOSE to an object

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtAttachVO(VirtContext VC, float mass, float *inertia)
```

DESCRIPTION

The function `virtAttachVO` implements an attachment of an object to the VIRTUOSE, and computes optimal control parameters while guaranteeing the global system stability.

The object is defined by the position of its center (reduction point of the force tensor), which has to be provided by using the `virtSetPosition` function before calling `virtAttachVO`.

This function is only accessible in control modes `COMMAND_TYPE_VIRTMECH`.

PARAMÈTERS

The parameter `VC` corresponds to the object of type `VirtContext` returned by the function `virtOpen`.

The parameter `mass` corresponds to the mass of the object, expressed in kg.

The parameter `inertia` corresponds to the inertia matrix of the object, stored in lines as a vector with 9 values.

RETURN VALUE

In case of success, the function `virtAttachVO` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtDetachVO`, `virtSetPosition`, `virtSetSpeed`, `virtGetForce`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 20

NAME

`virtClose` – closing of connection to the controller of the VIRTUOSE.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtClose(VirtContext VC);
```

DESCRIPTION

The function `virtClose` closes the connection to the controller of the VIRTUOSE.

PARAMÈTERS

The parameter `VC` corresponds to the object of type `VirtContext` returned by the function `virtOpen`.

RETURN VALUE

In case of success, the function `virtClose` returns `0`. Otherwise, it returns `-1` and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtOpen`, `virtGetErrorCode`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 21

NAME

virtDetachVO – detach an object from the VIRTUOSE

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtDetachVO(VirtContext VC)
```

DESCRIPTION

The function `virtDetachVO` cancels the attachment of an object with the VIRTUOSE, and set all control parameters to zero.

This function is only accessible in control modes `COMMAND_TYPE_VIRTMECH` and `COMMAND_TYPE_IMPEDANCE`.

PARAMÈTERS

The parameter `VC` corresponds to the object of type `VirtContext` returned by the function `virtOpen`.

RETURN VALUE

In case of success, the function `virtDetachVO` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtAttachVO`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 22

NAME

`virtDisplayHardwareStatus` – Hardware-level failure diagnosis.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtDisplayHardwareStatus (VirtContext VC, FILE *file)
```

DESCRIPTION

The function `virtDisplayHardwareStatus` returns a state vector of the VIRTUOSE hardware-level, in order to diagnose breakdowns and errors of each material component. The components considered are the amplifiers and the position encoders.

PARAMETERS

The parameter `VC` corresponds to the object of type `VirtContext` returned by the function `virtOpen`.

The parameter `file` is currently not used. The output device is defined by the function `virtSetOutputFile` (or `stdout` by default).

RETURN VALUE

In case of success, the function `virtDisplayHardwareStatus` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`VirtSetOutputFile`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 23

NAME

`virtGetBaseFrame` – Read the current position of the base reference frame

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetBaseFrame(VirtContext VC, float *pos)
```

DESCRIPTION

The function `virtGetBaseFrame` returns the current position of the base reference frame with respect to the observation reference frame.

PARAMETERS

The parameter `VC` corresponds to the object of type `VirtContext` returned by the function `virtOpen`.

The parameter `pos` corresponds to the transformation from the observation reference frame to the base reference frame.

RETURN VALUE

In case of success, the function `virtGetBaseFrame` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtSetBaseFrame`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 24

NAME

virtGetButton – Read the state of a push-button

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetButton(VirtContext VC, int button, int *state)
```

DESCRIPTION

The function `virtGetButton` returns the state of one of the push-buttons situated on the tool placed at the tip of the Virtuose.

PARAMETERS

The parameter `VC` corresponds to the object of type `VirtContext` returned by the function `virtOpen`.

The parameter `button` corresponds of the button index.

The parameter `state` is set by the function (0: button released, 1: button pushed).

RETURN VALUE

In case of success, the function `virtGetButton` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`VirtGetDeadMan`, `virtGetEmergencyStop`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 25

NAME

`virtGetCommandType` – Read the current control mode

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetCommandType(VirtContext VC, VirtCommandType *type)
```

DESCRIPTION

The function `virtGetCommandType` returns the current control mode.

PARAMÈTERS

The parameter `VC` corresponds to the object of type `VirtContext` returned by the function `virtOpen`.

The `type` parameter is set by the function.

Possible values are:

`COMMAND_TYPE_IMPEDANCE`

Force/position control

`COMMAND_TYPE_VIRTMECH`

Position/force control with virtual mechanism

RETURN VALUE

In case of success, the function `virtGetCommandType` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtSetCommandType`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 26

NAME

virtGetDeadMan – Read the state of the safety sensor

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetDeadMan(VirtContext VC, int *deadman)
```

DESCRIPTION

The function `virtGetDeadMan` returns the state of the safety sensor placed at the end of the Virtuose (also called "dead man sensor").

PARAMETERS

The parameter `VC` corresponds to the object of type `VirtContext` returned by the function `virtOpen`.

The parameter `deadman` is set by the function. A value of 1 means that the safety sensor is active (user present), a value of 0 that the sensor is inactive (user absent).

RETURN VALUE

In case of success, the function `virtGetDeadMan` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtGetEmergencyStop`, `virtGetPowerOn`

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 27

NAME

`virtGetEmergencyStop` – Read the state of the emergency stop.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetEmergencyStop(VirtContext VC, int *stop)
```

DESCRIPTION

The function `virtGetEmergencyStop` returns the state of the emergency stop.

PARAMETERS

The parameter `VC` corresponds to the object of type `VirtContext` returned by the function `virtOpen`.

The parameter `stop` is set by the function. A value of 1 means that the chain is closed (the system is operational), a value of 0 means that it is open (the system is stopped).

RETURN VALUE

In case of success, the function `virtGetEmergencyStop` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtGetDeadMan`, `virtGetPowerOn`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 28

NAME

virtGetError – Operational status of the Virtuose

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetError (VirtContext VC, int* error)
```

DESCRIPTION

The function `virtGetError` tests the operational status of the Virtuose, in terms of breakdowns or simple errors at the level of each material component. The components tested are the amplifiers and the position encoders.

PARAMETERS

The parameter `VC` corresponds to the object of type `VirtContext` returned by the function `virtOpen`.

The parameter `error` is set by the function. Value 0 means that the virtuose is in good operational conditions. Value 1 means breakdown or error of operation.

RETURN VALUE

In case of success, the function `virtGetError` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`


The parameter `VC` does not correspond to a valid `VirtContext` object.

`VIRT_E_HARDWARE_ERROR`

This error message means the presence of one or more breakdowns or errors.

SEE ALSO

`virtDisplayHardwareStatus`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 29

NAME

`virtGetErrorCode` – Get the code of the last error encountered.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetErrorCode(VirtContext VC);
```

DESCRIPTION

The function `virtGetErrorCode` returns the code of the last error encountered on the connection with the Virtuose controller corresponding to parameter VC.

PARAMÈTERS

The parameter VC corresponds to the VirtContext object returned by the function `virtOpen`, or NULL in the case of an error in the call to `virtOpen`.

RETURN VALUE

By definition, the function `virtGetErrorCode` always returns an error code (see below).

ERRORS

The errors below have a generic significance. Sometimes, the diagnostic depends on the function which signalled an error. Refer then to the description of the function.

VIRT_E_NO_ERROR

No error encountered.

VIRT_E_OUT_OF_MEMORY

Memory allocation error.

VIRT_E_INVALID_CONTEXT

An invalid `VirtContext` parameter was given to the API.

VIRT_E_COMMUNICATION_FAILURE

An error was encountered in the communication with the controller of the Virtuose.

VIRT_E_FILE_NOT_FOUND

The file given as parameter does not exist.

VIRT_E_WRONG_FORMAT

An error was detected in a format of data given to the API.

VIRT_E_TIME_OUT

The maximum time was exceeded in waiting for an answer from the controller of the Virtuose.

VIRT_E_NOT_IMPLEMENTED

The function is not implemented in this version of the API.


VIRT_E_VARIABLE_NOT_AVAILABLE

The required variable is not available on this model of Virtuose.

VIRT_E_INCORRECT_VALUE

A value transmitted to API is incorrect or outside of authorized bounds.


VIRT_E_HARDWARE_ERROR

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 30

A failure was detected in the operation of one hardware component of the Virtuose.

SEE ALSO

virtOpen, virtGetErrorMsg

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 31

NAME

`virtGetErrorMessage` – Get a description of an error code.

SYNOPSIS

```
#include "virtuoseAPI.h"
const char *virtGetErrorMessage(int code)
```

DESCRIPTION

The function `virtGetErrorMessage` returns a message describing the error code given as parameter.

PARAMETERS

The parameter `code` corresponds to the error code.

RETURN VALUE


In all the cases, the function `virtGetErrorMessage` returns an error message.

ERRORS

None

SEE ALSO

`virtGetErrorCode`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 32

NAME

`virtSetTimeStep` – Set the value of the simulation timestep.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetTimeStep(VirtContext VC, float timeStep)
```

DESCRIPTION

The function `virtSetTimeStep` informs the Virtuose controller of the simulation timestep. This value is used in order to guarantee the stability of the system. The function must be called before the selection of the type of control mode.

PARAMETERS

The parameter `VC` corresponds to the `VirtContext` object returned by the function `virtOpen`,
The parameter `timeStep` corresponds to the simulation timestep, expressed in seconds.

RETURN VALUE

In case of success, the function `virtGetError` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCORRECT_VALUE`

The value of the parameter `timeStep` is incorrect. This error is generated in the following cases:


- negative or null value
- value lower than the timestep of the controller
- value too large to ensure the stability of the system

`VIRT_E_CALL_TIME`

The function is called after the selection of the type of control mode.

SEE ALSO

`virtGetTimeStep`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 33

NAME

`virtGetTimeStep` – Read the current simulation timestep.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetTimeStep(VirtContext VC, float* step)
```

DESCRIPTION

The function `virtGetTimeStep` returns the value of the simulation timestep given beforehand by the function `virtGetTimeStep`. It returns zero if not defined.

PARAMETERS

The parameter `VC` corresponds to the `VirtContext` object returned by the function `virtOpen`,

The parameter `step` is set by the function, and corresponds of the simulation timestep expressed in seconds.

RETURNS VALUE

The function `virtGetTimeStep` returns `0` in all the cases.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetTimeStep`

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 34

NAME

`virtGetForce` – Return the force tensor to be applied to the attached object

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetForce(VirtContext VC, float *force_p_env_r_ps)
```

DESCRIPTION

The function `virtGetForce` returns the force tensor to be applied to the object attached to the VIRTUOSE, allowing the dynamic simulation of the scene.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `force_p_env_r_ps` is set by the function `virtGetForce`. It corresponds to the force applied by the user to the Virtuose, in the form of a 6 component force tensor. This force is expressed with respect to the center of the object, in the coordinates of the environment reference frame.

RETURN VALUE

In case of success, the function `virtGetError` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `vc` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtSetPosition`, `virtSetSpeed`, `virtAttachVO`

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 35

NAME

`virtGetForceFactor` – Read the current value of the force scale

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetForceFactor(VirtContext VC, float *factor)
```

DESCRIPTION

The function `virtGetForceFactor` returns the current value of the force factor, which corresponds to a scaling between the forces exerted at the tip of the VIRTUOSE and those computed in the simulation.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `factor` is set by the function `virtGetForceFactor`. A factor smaller than the 1 corresponds to an amplification of the forces exerted by the user at the tip of the VIRTUOSE towards the simulation.

RETURN VALUE

In case of success, the function `virtGetError` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtSetForceFactor`, `virtGetSpeedFactor`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 36

NAME

`virtGetIndexingMode` – Read the current indexing mode.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virt(VirtContext VC, VirtIndexingMode *mode)
```

DESCRIPTION

The function `virtGetIndexingMode` returns the current indexing mode (also called offset mode).

PARAMÈTERS

The parameter `VC` correspond to a `VirtContext` object returned by the `virtOpen` function.

The parameter `mode` is set by the function. The possible values are:

`INDEXING_ALL`

The indexing acts on the translations and rotations

`INDEXING_TRANS`

The indexing acts only on the translations.

`INDEXING_NONE`

The indexing is inoperative

RETURN VALUE

In case of success, the function `virtGetIndexingMode` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetIndexingMode`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 37

NAME

`virtGetObservationFrame` – Read the current position of the observation reference frame.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetObservationFrame(VirtContext VC, float *pos)
```

DESCRIPTION

The function `virtGetObservationFrame` returns the current position of the observation reference frame with respect to the environment reference frame.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `pos` corresponds to the transformation from the environment reference frame to the observation reference frame.

RETURN VALUE

In case of success, the function `virtGetObservationFrame` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetObservationFrame`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 38

NAME

`virtGetPosition` – Read the current position of the Virtuose, or of the object attached to the Virtuose

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetPosition(VirtContext VC, float *pos_p_env)
```

DESCRIPTION

The function `virtGetPosition` returns the current position of the Virtuose, or of the object attached to the VIRTUOSE if the function `virtAttachVO` has been called before.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `pos_p_env` is set by the function `virtGetPosition`. It is expressed in the form of a displacement vector with 7 components, in the coordinates of the environment reference frame.

RETURN VALUE

In case of success, the function `virtGetPosition` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetPosition`, `virtAttachVO`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 39

NAME

`virtGetPowerOn` – Read the state of the motor power supply.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetPowerOn(VirtContext VC, int *poweron)
```

DESCRIPTION

The function `virtGetPowerOn` returns the state of the motor power supply.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `poweron` is set by the function. A value of 1 means that the motors are under power, a value 0 that they are not.

RETURN VALUE

In case of success, the function `virtGetPowerOn` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetPowerOn`, `virtGetEmergencyStop`, `virtGetDeadMan`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 40

NAME

`virtGetSpeed` – Read the speed of the Virtuose, or of the object attached to the Virtuose.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetSpeed(VirtContext VC, float *speed_p_env_r_ps)
```

DESCRIPTION

The function `virtGetSpeed` returns the speed of the Virtuose, or of the object attached to the VIRTUOSE if the function `virtAttachVO` was called before.

PARAMÈTERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `speed_p_env_r_ps` is set by the function `virtGetSpeed`. It is expressed in the form of a cinematic tensor with 6 components, giving the speed of the Virtuose (or of the object attached to it) with respect to tip of the Virtuose (or to the center of the object attached to it) in the coordinates of the environment reference frame.

RETURN VALUE

In case of success, the function `virtGetSpeed` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetSpeed`, `virtAttachVO`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 41

NAME

`virtGetSpeedFactor` – Read the current value of the displacement scale factor.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetSpeedFactor(VirtContext VC, float *factor)
```

DESCRIPTION

The function `virtGetSpeedFactor` returns the current value of the speed factor, which corresponds to a scaling between movements of the VIRTUOSE and those in the simulation.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `factor` is set by the function `virtGetSpeedFactor`. A value larger than 1 corresponds to an expansion of the workspace of the Virtuose.

RETURN VALUE

In case of success, the function `virtGetSpeedFactor` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetSpeedFactor`, `virtGetForceFactor`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 42

NAME

`virtGetTimeLastUpdate` – Get the time elapsed since the last update of the Virtuose state vector.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetTimeLastUpdate (VirtContext, unsigned int *tout);
```

DESCRIPTION

The function `virtGetTimeLastUpdate` returns the time elapsed since the last update of the Virtuose state vector.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `tout` is set by the function with the value in CPU ticks of the time elapsed since the last update of the state vector received from the Virtuose controller.

RETURN VALUE


In case of success, the function `virtGetTimeLastUpdate` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 43

NAME

`virtGetTimeoutValue` – Read the current timeout value.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetTimeoutValue(VirtContext VC, float *tout)
```

DESCRIPTION

The function `virtGetTimeoutValue` returns the current value of the timeout for the communication with the Virtuose controller.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `tout` is set by the function with the current value of the timeout expressed in seconds.

RETURN VALUE

In case of success, the function `virtGetTimeoutValue` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetTimeoutValue`

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 44

NAME

`virtOpen` – Open a connection to the controller of the Virtuose

SYNOPSIS

```
#include "virtuoseAPI.h"
VirtContext virtOpen(const char *host)
```

DESCRIPTION

The function `virtOpen` opens a connection to the controller of the Virtuose.

PARAMETERS

The parameter `host` corresponds to the URL (*Uniform Ressource Locator*) of the VIRTUOSE controller to connect to.

In the current version of the API, only one type of communication protocol is available, therefore the URL is always in the form:

```
"udpdxdr://identification:port_number+interface"
```

`udpdxdr` is the only protocol available in the current version of the API.

`identification` should be replaced by the host name of the VIRTUOSE controller if it can be resolved by a DNS, or else by its IP address in dotted form (e.g. "192.168.0.1").

`port_number` should be replaced by the port number to be used by the API to connect to the Virtuose controller. The default value is 0, and in that case the API looks for a free port number starting from 3131.

`interface` designates the physical interface to be used by the API (ignored in the case of `udpdxdr`).

In case only `identification` is given, the URL is completed as follows:

```
"udpdxdr://identification:0"
```

Note: the automatic completion is limited to `udpdxdr` only.

The initial prefix "url:" defined in the URL standard is supported but ignored.

RETURN VALUE

In case of success, the function `virtOpen` returns a pointer of type `VirtContext` (pointer to a hidden data structure). Otherwise, it returns `NULL` and the function `virtGetErrorCode` gives access to an error code.

ERRORS


`VIRT_E_SYNTAX_ERROR`

The parameter `host` does not correspond to a valid URL.

`VIRT_E_COMMUNICATION_FAILURE`

The function `virtOpen` did not succeed in setting up a connection to the Virtuose.

`VIRT_E_OUT_OF_MEMORY`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 45

The function `virtOpen` could not allocate enough memory for the object `VirtContext`.

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the Virtuose controller is not compatible with that of the API.

SEE ALSO

`virtClose`, `virtGetErrorCode`

NAME

`virtSetCommandType` – Set the desired control mode

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetCommandType(VirtContext VC, VirtCommandType type)
```

DESCRIPTION

The function `virtSetCommandType` changes the control mode of the Virtuose device.

PARAMÈTERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `type` corresponds to the desired control mode. The possible values are:

`COMMAND_TYPE_NONE`

No possible movement

`COMMAND_TYPE_IMPEDANCE`

Force/position control

`COMMAND_TYPE_VIRTMECH`

Position/force control with virtual mechanism

RETURN VALUE

In case of success, the function `virtSetCommandType` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`


The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCORRECT_VALUE`

The value of parameter `type` does not correspond to a valid control mode.

SEE ALSO

`virtGetCommandType`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 47

NAME

`virtSetDebugFlags` – Set the level of diagnosis information output.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetDebugFlags(VirtContext VC, unsigned short flags)
```

DESCRIPTION

The function `virtSetDebugFlags` sets the level of diagnosis information output generated by the VIRTUOSE API.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `flags` is a field of bits corresponding to the various levels of diagnosis to activate:

- `DEBUG_SERVO`: information related to the system control stability
- `DEBUG_LOOP`: information related to the simulation timing

RETURN VALUE


In case of success, the function `virtSetDebugFlags` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 48

NAME

`virtSetForce` – Send the force to be applied to the VIRTUOSE

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetForce(VirtContext VC, float *force_p_env_r_ps)
```

DESCRIPTION

The function `virtSetForce` sends the force to be applied to the VIRTUOSE , in the case of a force/position control (impedance control).

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `force_p_env_r_ps` contains the force to be applied. It is expressed in the form of a dynamic tensor with 6 components, with respect to the Virtuose end-effector and expressed in the coordinates of the environment reference frame.

This function is accessible only in `COMMAND_TYPE_IMPEDANCE` control mode.

RETURN VALUE

In case of success, the function `virtSetForce` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetCommandType`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 49

NAME

`virtSetForceFactor` – Set the force scale factor.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetForceFactor(VirtContext VC, float factor)
```

DESCRIPTION

The function `virtSetForceFactor` sets the value of the force factor, which corresponds to a scaling between the forces exerted at the tip of the VIRTUOSE and those computed in the simulation.

The function must be called before the selection of the control mode.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `factor` corresponds to the force scale. A value smaller than 1 corresponds to an amplification of the forces from the Virtuose towards the simulation.

RETURN VALUE

In case of success, the function `virtSetForceFactor` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCORRECT_VALUE`

The value of the parameter `factor` is incorrect. This error is generated in the following cases:

- `factor` is negative or null
- speed factor too large or too small to guarantee the stability of the system

`VIRT_E_CALL_TIME`

The function is called after the selection of the control mode.

SEE ALSO

`virtGetForceFactor`, `virtSetSpeedFactor`

NAME

`virtSetIndexingMode` – Modify the mode of indexing

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetIndexingMode(VirtContext VC, VirtIndexingType mode)
```

DESCRIPTION

The function `virtSetIndexingMode` changes the mode of indexing (also called offset).

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `mode` corresponds to the mode of indexing. The different values are:

`INDEXING_ALL`

Indexing is active for both translation and rotation movements, whenever the offset button is pushed or the power is off.

`INDEXING_TRANS`

Indexing is active only on the translation movements. When power is turned on, the device is constrained along a line segment going back to the orientation it had before switching off.

`INDEXING_NONE`

Indexing is inactive. When power is turned on, the device is constrained along a line segment going back to the position it had before switching off.

Other values are implemented, which correspond to the same modes but force-feedback is inhibited during indexing.

RETURN VALUE

In case of success, the function `virtSetIndexingMode` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`


The parameter `VC` does not correspond to a valid `VirtContext` object.

`VIRT_E_INCORRECT_VALUE`

The value of parameter `mode` does not correspond to a valid mod.

SEE ALSO

`virtGetIndexingMode`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 51

NAME

`virtSetObservationFrame` – Move the observation frame

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetObservationFrame(VirtContext VC, float *pos)
```

DESCRIPTION

The function `virtSetObservationFrame` modifies the position of the observation frame with respect to the reference of environment.
It can be called at any time.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `pos` corresponds to the transformation from the environment reference frame to the observation reference frame, expressed as a displacement vector of 7 components.

RETURN VALUE

In case of success, the function `virtSetObservationFrame` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`


The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCORRECT_VALUE`

The parameter `pos` is not a valid displacement vector.

SEE ALSO

`VirtGetObservationFrame`
`VirtSetObservationFrameSpeed`

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 52

NAME

`virtSetObservationFrameSpeed` – Modify the speed of the observation reference frame

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetObservationFrameSpeed(VirtContext VC, float *speed)
```

DESCRIPTION

The function `virtSetObservationFrameSpeed` sets the speed of the observation reference frame with respect to the environment reference frame. It can be called at any time.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `speed` corresponds to the speed of motion of the observation reference frame with respect to the environment reference frame.

RETURN VALUE

In case of success, the function `virtSetObservationFrameSpeed` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtGetObservationFrame`

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 53

NAME

`virtSetPosition` – Modify the current control position.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetPosition(VirtContext VC, float *pos_p_env)
```

DESCRIPTION

The function `virtSetPosition` modifies the current value of the control position and sends it to the Virtuose controller. If an object is attached to the Virtuose (`virtAttachVO` called before), then the control point is the center of the object, otherwise it is the center of the Virtuose end-effector.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `pos_p_env` is a displacement vector of 7 components (see glossary).

RETURN VALUE

In case of success, the function `virtSetPosition` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object


`VIRT_E_INCORRECT_VALUE`

The value of the parameter `pos_p_env` is incorrect. This error is generated in the following case:

- The module of the quaternion part is different from 1.0

SEE ALSO

`VirtAttachVO`, `virtSetSpeed`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 54

NAME

virtSetSpeed – Modify the current control speed

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetSpeed(VirtContext VC, float *speed_p_env_r_ps)
```

DESCRIPTION

The function `virtSetSpeed` modifies the current value of the control speed. If an object is attached to the Virtuose (`virtAttachVO` called before), then the control point is the center of the object, otherwise it is the center of the Virtuose end-effector.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `speed_p_env_r_ps` corresponds to the speed with respect to the control point expressed in the coordinates of the environment reference frame.

RETURN VALUE

In case of success, the function `virtSetSpeed` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtAttachVO`, `virtSetPosition`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 55

NAME

`virtSetSpeedFactor` – Modify the movement scale factor.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetSpeedFactor(VirtContext VC, float factor)
```

DESCRIPTION

The function `virtSetSpeedFactor` modifies the speed factor, which corresponds to a scaling of the workspace of the haptic device.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `factor` corresponds to the scale factor to be applied. A value larger than 1.0 means that the movements of the Virtuose are amplified inside the simulation.

RETURN VALUE

In case of success, the function `virtSetSpeedFactor` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object;


`VIRT_E_INCORRECT_VALUE`

The value of the parameter `factor` is incorrect; this error is generated in the following cases:

- speed factor is negative or null
- speed factor is too large or too small to guarantee the stability of the system

SEE ALSO

`virtGetSpeedFactor`, `virtSetForceFactor`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 56

NAME

`virtSetTimeoutValue` – Modify the communication timeout value

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetTimeoutValue(VirtContext VC, float tout)
```

DESCRIPTION

The function `virtSetTimeoutValue` modifies the timeout value used in communications with the Virtuose controller.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `tout` corresponds to the new timeout value expressed in seconds.

RETURN VALUE

In case of success, the function `virtSetTimeoutValue` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCORRECT_VALUE`

The value of parameter `tout` is incorrect.

SEE ALSO

`virtGetTimeoutValue`

NAME

`virtVmSetType`— Select one type of virtual mechanism.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtVmSetType(VirtContext VC, virtVmType type)
```

DESCRIPTION

The function `virtVmSetType` selects the type of virtual mechanism to be used. This function is only available in `COMMAND_TYPE_VIRTMECH` control mode.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `type` corresponds to the type of virtual mechanism. The available types are:

`VM_TYPE_CartMotion`

Virtual guide with 1 degree of freedom, which corresponds to a line segment (in 6 degrees-of-freedom), joining the current position of the end-effector to the center of the virtual mechanism base frame.

`VM_TYPE_Spline`

Virtual guide with 1 degree of freedom, based on a 6D spline curve.

`VM_TYPE_Rx`

Virtual guide with 1 degree of freedom, i.e. one rotation around the X axis of the virtual mechanism base frame.

`VM_TYPE_Tx`

Virtual guide with 1 degree of freedom, i.e. one translation along the X axis of the virtual mechanism base frame.

`VM_TYPE_TxTy`

Virtual guide with 2 degrees of freedom, corresponding to two translations along axes X and Y of the virtual mechanism base frame.

`VM_TYPE_TxRx`

Virtual guide with 2 degrees of freedom, corresponding to one translation and one rotation around axis X of the virtual mechanism base frame.

`VM_TYPE_RxRyRz`


Virtual guide with 3 degrees of freedom, corresponding to three rotations around the center of the virtual mechanism base frame.

`VM_TYPE_Crank`

Virtual guide with 2 degrees of freedom, corresponding to the movement of a crank around axis X of the virtual mechanism base frame.

RETURN VALUE

In case of success, the function `virtVmSetType` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 58

ERRORS

VIRT_E_INVALID_CONTEXT


The parameter `VC` does not correspond to a valid `VirtContext` object

VIRT_E_INCORRECT_VALUE

The value of parameter `type` is incorrect.

SEE ALSO

`virtVmSetBaseFrame`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 59

NAME

`virtVmActivate` – Activate the selected type of virtual mechanism.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtVmActivate(VirtContext VC)
```

DESCRIPTION

The function `virtVmActivate` activates the virtual mechanism selected before with `virtVmSetType`.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `virtVmActivate` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtVmDeactivate`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 60

NAME

virtVmDeactivate – Deactivate the current virtual mechanism

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtVmDeactivate(VirtContext VC)
```

DESCRIPTION

The function `virtVmDeactivate` deactivates the virtual mechanism previously activated.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `virtVmDeactivate` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtVmActivate`

NAME

`virtVmSetSamplingTimeStep` – Set the value of the trajectory sampling period.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtVmSetSamplingTimeStep (VirtContext VC, float timeStep
                               unsigned int* recordTime)
```

DESCRIPTION

The function `virtVmSetSamplingTimeStep` sets the value of the trajectory sampling period and maximum recording time, to be used in conjunction of functions `virtTrajRecordStart` and `virtTrajRecordStop`.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `timeStep` corresponds to the sampling period expressed in seconds.

The parameter `recordTime` corresponds to the maximum recording time expressed by seconds.

RETURN VALUE

In case of success, the function `VirtVmSetSamplingTimeStep` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object


`VIRT_E_INCORRECT_VALUE`

The value of one parameter is incorrect. This error is generated by these following cases:

- Value is negative or null
- Value is smaller than the timestep of the Vrtuose controller

SEE ALSO

`virtTrajRecordStart`, `virtTrajRecordStop`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 62

NAME

`virtTrajRecordStart` – Start recording the trajectory

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtTrajRecordStart (VirtContext VC)
```

DESCRIPTION

The function `virtTrajRecordStart` starts the recording of the trajectory.

PARAMÈTERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `virtTrajRecordStart` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtTrajRecordStop`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 63

NAME

virtTrajRecordStop – Stop recording the trajectory

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtTrajRecordStop (VirtContext VC)
```

DESCRIPTION

The function `virtTrajRecordStop` stops the recording of the trajectory started by `virtTrajRecordStart`.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `virtTrajRecordStop` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtTrajRecordStart`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 64

NAME

`virtVmStartTrajSampling` – Initialize the transfer of the recorded trajectory.

SYNOPSIS

```
#include "virtuoseAPI.h"
int  virtVmStartTrajSampling  (VirtContext  VC,  unsigned  int
nbSamples)
```

DESCRIPTION

The function `virtVmStartTrajSampling` starts the transfer of the control points of a trajectory previously recorded.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `nbSamples` corresponds to the number of points requested. In case the number of points is lower than the total number of recorded points, a new sampling is done in order to fit to the requested number of control points.

RETURN VALUE

In case of success, the function `VirtVmStartTrajSampling` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`


The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCORRECT_VALUE`

The parameter `nbSamples` is negative or higher than 3000 points.

SEE ALSO

`virtVmGetTrajSamples`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 65

NAME

`virtVmGetTrajSamples` – Transfer the trajectory control points

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtVmGetTrajSamples (VirtContext VC, float* samples)
```

DESCRIPTION

The `virtVmGetTrajSamples` function transfers the control points of a previously recorder trajectory. This function blocks until the end of the transfer of the control points from the Virtuose controller

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `samples` corresponds to a pre-allocated array of floats of size `7*nbSamples`, where `nbSamples` is the number of control points specified with the function `virtVmStartTrajSampling`, each control point being defined as a displacement vector of 7 components.

RETURN VALUE

In case of success, the function `virtVmGetTrajSamples` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtVmStartTrajSampling`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 66

NAME

`virtGetPhysicalPosition` – Read the physical position of the haptic device.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetPhysicalPosition (VirtContext VC, float* pos)
```

DESCRIPTION

The function `virtGetPhysicalPosition` returns the physical position of the Virtuose with respect to its base.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The `pos` function corresponds to the position. It is expressed in the coordinates of the base reference frame of the Virtuose.

RETURN VALUE

In case of success, the function `virtGetPhysicalPosition` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetBaseFrame`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 67

NAME

`virtGetAvatarPosition` – Read the indexed position of the end-effector.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetAvatarPosition (VirtContext VC, float* pos)
```

DESCRIPTION

The function `VirtGetAvatarPosition` returns the current position of the end-effector, taking into account the current offsets (or indexing) and the motion scale factor.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The `pos` parameter corresponds to the position. It is expressed as a displacement vector of 7 components, in the coordinates of the environment reference frame. In case no object has been attached to the Virtuose (no previous call to `virtAttachVO`), the function returns the same result as `virtGetPosition`.

RETURN VALUE

In case of success, the function `virtGetAvatarPosition` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetSpeedFactor`, `virtGetPosition`

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 68

NAME

`virtVmSetDefaultToCartesianPosition` – Set the default behaviour to blocked mode.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtVmDefaultToCartesianPosition (VirtContext VC)
```

DESCRIPTION

The function `virtVmDefaultToCartesianPosition` sets the default behaviour of the haptic device to blocked. When no virtual guide is active, the haptic device cannot be moved. This function is only available in `COMMAND_TYPE_VIRTMECH` control mode.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `VirtVmDefaultToCartesianPosition` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetDefaultToTransparentMode`
`virtVmActivate`
`virtVmDeactivate`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 69

NAME

`virtVmSetDefaultToTransparentMode` – Set the default behaviour to free motion.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtVmDefaultToTransparentMode (VirtContext VC)
```

DESCRIPTION

The function `virtVmDefaultToTransparentMode` sets the default behaviour of the haptic device to free motion. When no virtual guide is active, the haptic device can be moved freely. This function is only available in `COMMAND_TYPE_VIRTMECH` control mode.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `virtVmDefaultToTransparentMode` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetDefaultToCartesianMode`
`virtVmActivate`
`virtVmDeactivate`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 70

NAME

`virtVmSetBaseFrame` – Modify the position of the virtual mechanism base frame

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtVmSetBaseFrame (VirtContext VC, float *base)
```

DESCRIPTION

The function `virtVmSetBaseFrame` changes the position of the virtual mechanism base frame.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `base` corresponds to the transformation from the environment reference frame to the virtual mechanism base frame, expressed as a displacement vector of 7 components (see glossary).

RETURN VALUE

In case of success, the function `VirtVmSetBaseFrame` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object.

SEE ALSO

`virtVmType`
`virtVmGetBaseFrame`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 71

NAME

`virtVmGetBaseFrame` – Read the current position of the virtual mechanism base frame.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtVmGetBaseFrame (VirtContext VC, float *base)
```

DESCRIPTION

The function `VirtVmGetBaseFrame` returns the current position of the virtual mechanism base frame.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `base` corresponds to the reference frame of the base. It is expressed compared to the reference frame of environment.

RETURN VALUE

In case of success, the function `VirtVmGetBaseFrame` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object;

SEE ALSO

`virtVmSetBaseFrame`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 72

NAME

virtConvertRGBToGrayscale – Convert a RGB colour to greyscale.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtConvertRGBToGrayscale (VirtContext VC, float *rgb,
float* gray)
```

DESCRIPTION

The function `virtConvertRGBToGrayscale` converts a colour defined as (red, green, blue) values to a greyscale colour. The following ratios are used:

- 29.9% of the red value,
- 58.7% of the green value,
- 11.4% of the blue value.

PARAMÈTERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `rgb` is a table of three components (varying between 0 and 1), of which the first corresponds to the red color, the second with the green color and the third with the blue colour.

The parameter `gray` is a single floating point value, set by the function.

RETURN VALUE

In case of success, the function `virtConvertRGBToGrayscale` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object;

SEE ALSO

`virtVmSetTexture`

NAME

`virtSetTexture` – Simulate a haptic texture.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetTexture (VirtContext VC, float *H_texture_p_env,
float* intensity, int reinit)
```

DESCRIPTION

The function `virtSetTexture` simulates a haptic texture, related to a gradient in a greyscale value. A haptic texture gives an impression of surface roughness, without actually modifying the geometry. Light colours are rendered as bumps, and dark as holes.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `H_texture_p_env` corresponds to the position of the contact point with the surface, expressed as a displacement vector of 7 components in the coordinates of the environment reference frame.

The parameter `intensity` corresponds to the greyscale value at the contact point.

The parameter `reinit` cancels the current texture rendering. It should be set to 1 once when entering contact, and then to 0 for texture rendering.

RETURN VALUE

In case of success, the function `virtSetTexture` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`


The parameter `VC` does not correspond to a valid `VirtContext` object;

`VIRT_E_INCORRECT_VALUE`

The parameter `intensity` is not a value between 0 and 1.

SEE ALSO

`virtConvertRGBToGrayscale`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 74

NAME

`virtSetTextureForce` – Apply a force simulating a haptic texture.

SYNOPSIS

```
#include "virtuoseAPI.h"
int      virtSetTextureForce      (VirtContext      VC,      float
      *W_texture_p_env_r_ps)
```

DESCRIPTION

The function `virtSetTextureForce` adds a force directly at the end-effector of the haptic device. It is commonly used for rendering haptic textures or force fields, which cannot be rendered by the function `virtSetTexture`.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `W_texture_p_env_r_ps` corresponds to the force to be added, expressed as a dynamic tensor of 6 components with respect to the Virtuose end-effector, in the coordinates of the environment reference frame.


RETURN VALUE

In case of success, the function `virtSetTextureForce` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object;

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 75

NAME

`virtVmWaitUpperBound` – Wait until the upper bound of a virtual guide has been reached.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtVmWaitUpperBound (VirtContext VC)
```

DESCRIPTION

The function `VirtVmWaitUpperBound` is available only in control mode `COMMAND_TYPE_VIRTMECH` and for virtual guides of type `VM_TYPE_CartMotion` and `VM_TYPE_Spline`.

When a virtual guide is active, the function blocks until the upper bound of the virtual guide is reached, or the virtual guide is deactivated.

A typical use of this function is to execute trajectories defined as sequences of virtual guides.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `virtVmWaitUpperBound` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCORRECT_VM_TYPE`

The type of virtual guide is incorrect.

`VIRT_E_WRONG_MODE`


The type of control mode is incorrect.

`VIRT_E_CALL_TIME`

The function has been called before the activation of the virtual guide.

SEE ALSO

`virtVmSetType`
`virtVmActivate`
`virtVmDeactivate`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 76

NAME

`virtDisableControlConnexion` – Disable the watchdog control on the communication with the Virtuose controller

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtDisableControlConnexion (VirtContext VC, int disable)
```

DESCRIPTION

The function `virtDisableControlConnexion` disables (or enables) the watchdog control on the communication with the Virtuose controller. The role of the watchdog control is to stop force-feedback in case of a software failure on the simulation side. In practice, if the simulation does not update the control values during a time period of 2 seconds, the Virtuose considers that the simulation is dead and cuts off the force-feedback and the connection to the API.

For debugging purposes, it can be useful to disable the watchdog control, so that the simulation can be executed step-by-step without losing the connection with the Virtuose. In that case, it is essential to call the function `virtClose` at the end of the simulation, otherwise the haptic device will not be reset, and it could be impossible to establish a new connection.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `disable` should be set to 1 to disable the watchdog control, and to 0 to re-enable it.

RETURN VALUE

In case of success, the function `virtDisableControlConnexion` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtClose`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 77

NAME

virtSetPeriodicFunction – Time the simulation

SYNOPTISIS

```
#include "virtuoseAPI.h"
int virtSetPeriodicFunction (VirtContext VC,
                             void (*fn) (VirtContext, void*),
                             float* period,
                             void* arg);
```

DESCRIPTION

The function `virtSetPeriodicFunction` defines a callback function to be called at a fixed period of time, as timing for the simulation. The callback function is synchronized with messages received from the Virtuose controller, which arrive at very constant time intervals, and generate hardware interrupts not to be ignored by the operating system. In practise, this function is much more efficient for timing the simulation than common software timers.

PARAMÈTERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `fn` is a pointer to the callback function.

The parameter `period` corresponds the timing period, given in seconds. It should always be identical to the timestep given to the function `virtSetTimeStep`.

The parameter `arg` corresponds to an arbitrary parameter, to be used by the callback function.

RETURN VALUE

In case of success, the function `virtSetPeriodicFunction` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtStartLoop`
`virtStopLoop`
`virtSetTimeStep`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 78

NAME

`virtStartLoop` – Start the simulation timing.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtStartLoop (VirtContext VC);
```

DESCRIPTION

The function `virtStartLoop` starts the periodic loop, which executes the callback function defined with `VirtSetPeriodicFunction` at a fixed rate.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `virtStartLoop` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetPeriodicFunction`
`virtStopLoop`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 79

NAME

`virtStopLoop` – Stop the simulation timing.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtStopLoop (VirtContext VC);
```

DESCRIPTION

The function `virtStopLoop` stops the periodic loop, which executes the callback function defined with `VirtSetPeriodicFunction` at a fixed rate.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `virtStopLoop` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetPeriodicFunction`
`virtStartLoop`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 80

NAME

`virtIsInBounds` – Test whether the mechanical limits of the device workspace have been reached.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtIsInBounds (VirtContext VC, unsigned int *bounds);
```

DESCRIPTION

The function `VirtIsInBounds` returns a bit field giving the status of all mechanical limits of the device workspace.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `bounds` is set by the function, as a field of bit with the following meaning:

- `VIRT_BOUND_LEFT_AXE_1` corresponds to the left bound of axis 1,
- `VIRT_BOUND_RIGHT_AXE_1` corresponds to the right bound of axis 1,
- `VIRT_BOUND_SUP_AXE_2` corresponds to the upper bound of axis 2,
- `VIRT_BOUND_INF_AXE_2` corresponds to the lower bound of axis 2,
- `VIRT_BOUND_SUP_AXE_3` corresponds to the upper bound of axis 3,
- `VIRT_BOUND_INF_AXE_3` corresponds to the lower bound of axis 3,
- `VIRT_BOUND_LEFT_AXE_4` corresponds to the left bound of axis 4,
- `VIRT_BOUND_RIGHT_AXE_4` corresponds to the right bound of axis 4,
- `VIRT_BOUND_SUP_AXE_5` corresponds to the upper bound of axis 5,
- `VIRT_BOUND_INF_AXE_5` corresponds to the lower bound of axis 5,
- `VIRT_BOUND_LEFT_AXE_6` corresponds to the left bound of axis 6,
- `VIRT_BOUND_RIGHT_AXE_6` corresponds to the right bound of axis 6.


RETURN VALUE

In case of success, the function `virtIsInBounds` returns 0. Otherwise, it returns – 1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 81

NAME

`virtGetAlarm` – Read the current alarm status.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetAlarm (VirtContext VC, unsigned int *alarm);
```

DESCRIPTION

The function `virtGetAlarm` returns the current alarm status.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `alarm` is set by the function, as a field of bits with the following meaning:

- `VIRT_ALARM_OVERHEAT` means that one motor is overheated. Force-feedback is automatically reduced, until the motor has cooled down to an acceptable temperature.
- `VIRT_ALARM_SATURATE` means that the motor currents have reached their maximum and are saturated.
- `VIRT_ALARM_CALLBACK_OVERRUN` means that the execution time of the callback function defined with `virtSetPeriodicFunction` is greater than the timestep value. In that case, the real-time execution of the simulation cannot be guaranteed.

RETURN VALUE

In case of success, the function `virtGetAlarm` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSaturateTorque`
`virtSetPeriodicFunction`
`virtSetTimeStep`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 82

NAME

`virtActiveSpeedControl` – Activate hybrid force/speed control in translation.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtActiveSpeedControl (VirtContext VC,
                           float radius,
                           float speedFactor);
```

DESCRIPTION

The function `virtActiveSpeedControl` activates a hybrid control mode, with a standard position/force control inside a sphere, and a simple speed control (similar to joystick control) outside the sphere.

The sphere is defined by its center, which is always identical to the center of the device workspace, and its radius, which can be specified by the user.

Inside the sphere, the control mode is unchanged.

Outside the sphere, the positions given by `virtGetPosition` and `virtGetAvatarPosition` vary proportionally to the distance to the sphere affected by the parameter `speedFactor`, in the same directions, while a constant force tends to bring back the end-effector of the Virtuose inside the sphere.

The typical values of the parameter `radius` in meters are 0.1 for a device of type Virtuose 3D15-25, and 0.2 for a Virtuose 6D35-45.

The default value for `speedFactor` is 1.0.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `radius` corresponds to the radius of the sphere, given in meters.

The parameter `speedFactor` corresponds to a scaling of the speed.

RETURN VALUE

In case of success, the function `virtActiveSpeedControl` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`


The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCORRECT_VALUE`

The value of one of the two numeric parameters is negative or null.

SEE ALSO

`virtDeactiveSpeedControl`

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 83

NAME

`virtDeactiveSpeedControl` – Deactivate hybrid position/speed control in translation.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtDeactiveSpeedControl (VirtContext VC);
```

DESCRIPTION

The function `virtDeactiveSpeedControl` leaves the hybrid position/speed control mode, previously activated by `virtActiveSpeedControl`.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `virtDeactiveSpeedControl` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtActiveSpeedControl`

NAME

`virtSetCatchFrame` – Modify the attachment point of a virtual object

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtSetCatchFrame (VirtContext VC, float* frame);
```

DESCRIPTION

The function `virtSetCatchFrame` specifies the position of the point where an object can be attached to the Virtuose, also called “catch frame”. The catch frame has to be defined before calling `virtAttachVO`. It is reset to identity after each call to `virtDettachVO`. The rotation part of the catch frame position is not taken into account.

This function is only available in control mode `COMMAND_TYPE_VIRTMECH`.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `frame` corresponds to the transformation from the center of the virtual object to the center of the catch frame, expressed as a displacement vector of 7 components, only the first 3 of which are used.

RETURN VALUE

In case of success, the function `virtSetCatchFrame` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`


The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCORRECT_VALUE`

The value of the `frame` parameter is invalid.

SEE ALSO

`virtAttachVO`
`virtDettachVO`
`virtGetCatchFrame`

	VIRTUOSE API V3.60		HAP/02/RT.006	
	DOCUMENTATION		Rev. 1	page 85

NAME

`virtGetCatchFrame` – Read the current position of the attachment point

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetCatchFrame (VirtContext VC, float* frame);
```

DESCRIPTION

The function `virtGetCatchFrame` returns the current value of the catch frame, previously specified by `virtSetCatchFrame`.

The function is available only in control mode `COMMAND_TYPE_VIRTMECH`.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `frame` corresponds to the transformation from the center of the virtual object to the center of the catch frame, expressed as a displacement vector of 7 components, only the first 3 of which are used.

RETURN VALUE

In case of success, the function `virtGetCatchFrame` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtAttachVO`
`virtDettachVO`
`virtSetCatchFrame`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 86

NAME

`virtIsInShiftPosition` – Test the status of indexing.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtIsInShiftPosition(VirtContext VC, int* indexing);
```

DESCRIPTION

The function `virtIsInShiftPosition` returns the current state of indexing, that is:

- a value of 1 if the offset push-button is pressed or the power is off
- a value of 0 otherwise.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `indexing` corresponds to the state of indexing.

RETURN VALUE

In case of success, the function `virtIsInShiftPosition` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtSetIndexingMode`, `virtGetIndexingMode`, `virtGetPowerOn`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 87

NAME

`virtGetMouseState` – Read the state of push-buttons in mouse mode

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetMouseState (VirtContext VC,
                      int* active,
                      int* left,
                      int* right) ;
```

DESCRIPTION

The function `virtGetMouseState` returns the status of the mouse mode of the Virtuose device, as well as the state of the mouse push-buttons.

This function should be used when the mouse mode of the API cannot work properly, as in the following cases:

- The mouse events are not supported for your Operating System (e.g. IRIX and Linux).
- The mouse events are not to be used on the computer running the API.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `active` is set by the function (1 if the mouse mode is active, 0 otherwise).

The parameter `left` is set by the function with the state of the left button.

The parameter `right` is set by the function with the state of the right button.

RETURN VALUE

In case of success, the function `virtGetMouseState` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

NAME

`virtConvertDeplToHomogeneMatrix` – Convert a displacement vector into a matrix transform.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtConvertDeplToHomogeneMatrix (VirtContext VC,
                                     float* d,
                                     float* m);
```

DESCRIPTION

The function `virtConvertDeplToHomogeneMatrix` converts a displacement vector given as 7 components (see glossary) into a 4x4 matrix transform stored in columns:

a0	a4	a8	a12
a1	a5	a9	a13
a2	a6	a10	a14
a3	a7	a11	a15

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `d` is the displacement vector to convert.

The parameter `m` is the matrix transform, to be filled by the function.

RETURN VALUE

In case of success, the function `virtConvertDeplToHomogeneMatrix` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtConvertHomogeneMatrixToDepl`

NAME

`virtConvertHomogeneMatrixToDepl` – Convert a matrix transform into a displacement vector.

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtConvertHomogeneMatrixToDepl (VirtContext VC,
                                     float* d,
                                     float* m);
```

DESCRIPTION

The function `virtConvertHomogeneMatrixToDepl` converts a 4x4 matrix transform into a displacement vector (see glossary). The matrix transform should be stored in columns:

a0	a4	a8	a12
a1	a5	a9	a13
a2	a6	a10	a14
a3	a7	a11	a15

PARAMÈTERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `m` is the matrix transform to convert.

The parameter `d` is the displacement vector to be filled by the function.

RETURN VALUE

In case of success, the function `virtConvertHomogeneMatrixToDepl` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

SEE ALSO

`virtConvertDeplToHomogeneMatrix`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 90

NAME

virtGetTrackball – read the trackball

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetTrackball(VirtContext VC, int* x_move, int* y_move)
```

DESCRIPTION

La fonction `virtGetTrackball` allow to read the movements x and y of the trackball. Both positions are filtered into the controler.

PARAMÈTERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `x_move` corresponds to the position on x axis.


The parameter `y_move` corresponds to the position on y axis.

RETURN VALUE

In case of success, the function `virtGetTrackball` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 91

NAME

virtGetTrackballButton – read the buttons of the trackball

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtGetTrackballButton(VirtContext VC,
                           int* actif,
                           int* btn_gauche,
                           int* btn_milieu,
                           int* btn_droit)
```

DESCRIPTION

They are two operating modes of the handle : the Virtuose mode and the mouse mode. The mode changes with the switch located on the handle. In mouse mode, the function `virtGetTrackballButton` allow to read left, medium and right buttons. In Virtuose mode, all buttons are inactive.

PARAMÈTERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `btn_gauche` corresponds to the state of the left button.

The parameter `btn_milieu` corresponds to the state of the medium button.


The parameter `btn_droit` corresponds to the state of the right button.

RETURN VALUE

In case of success, the function `virtGetTrackballButton` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 92

NAME

`virtAddForce` – Add a force to the VIRTUOSE

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtAddForce(VirtContext VC, float *force_p_bm_r_pm)
```

DESCRIPTION

The function `virtAddForce` add a force to be applied to the VIRTUOSE.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `force_p_bm_r_pm` contains the force to be applied. It is expressed in the form of a dynamic tensor with 6 components, with respect to the Virtuose end-effector and expressed in the coordinates of the base frame.


RETURN VALUE

In case of success, the function `virtAddForce` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 93

NOM

`virtGetArticularPositionOfAdditionalAxe` – read articular position of the gripper

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int    virtGetArticularPositionOfAdditionalAxe(VirtContext  VC,
float* pos);
```

DESCRIPTION

This fonction read the articular position of an additional motor or of the gripper.

PARAMETRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `pos` corresponds to the position.


VALEUR DE RETOUR

In case of success, the function `virtGetArticularPositionOfAdditionalAxe` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 94

NOM

`virtSetArticularPositionOfAdditionalAxe` – send a consigne of articular position

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetArticularPositionOfAdditionalAxe(VirtContext VC,
float* pos);
```

DESCRIPTION

This fonction send a consigne of articular position to an additional motor or to the gripper.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `pos` corresponds to the position.


VALEUR DE RETOUR

In case of success, the function `virtGetArticularPositionOfAdditionalAxe` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 95

NOM

virtGetArticularSpeedOfAdditionalAxe – read articular speed of the gripper

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int      virtGetArticularSpeedOfAdditionalAxe(VirtContext    VC,
float* pos);
```

DESCRIPTION

This fonction read the articular speed of an additional motor or of the gripper.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `speed` corresponds to the speed.


VALEUR DE RETOUR

In case of success, the function `virtGetArticularSpeedOfAdditionalAxe` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 96

NOM

virtSetArticularSpeedOfAdditionalAxe – send a consigne of articular speed

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int      virtSetArticularSpeedOfAdditionalAxe(VirtContext    VC,
float* pos);
```

DESCRIPTION

This fonction send a consigne of articular speed to an additional motor or to the gripper.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `speed` corresponds to the speed.


VALEUR DE RETOUR

In case of success, the function `virtSetArticularSpeedOfAdditionalAxe` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 97

NOM

`virtSetArticularForceOfAdditionalAxe` – send a consigne of articular force

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int      virtSetArticularForceOfAdditionalAxe(VirtContext  VC,
float* effort);
```

DESCRIPTION

This fonction send a consigne of articular force to an additional motor or to the gripper.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `effort` corresponds to the articular force.


VALEUR DE RETOUR

In case of success, the function `virtSetArticularForceOfAdditionalAxe` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 98

NOM

virtGetArticularPosition – read articular position

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetArticularPosition(VirtContext VC, float* pos);
```

DESCRIPTION

This fonction read the articular position of the virtuose.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `pos` corresponds to the position. It is an array of float, with length the number of axes of the virtuose.

VALEUR DE RETOUR

In case of success, the function `virtGetArticularPosition` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the embedded software is smaller than 3.60

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 99

NOM

virtSetArticularPosition – send a consigne of articular position

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetArticularPosition(VirtContext VC, float* pos);
```

DESCRIPTION

This fonction send a consigne of articular position to the virtuose.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `pos` corresponds to the position. It is an array of float, with length the number of axes of the virtuose.

VALEUR DE RETOUR

In case of success, the function `virtSetArticularPosition` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the embedded software is smaller than 3.60

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 100

NOM

virtGetArticularSpeed – read articular speed

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetArticularSpeed(VirtContext VC, float* speed);
```

DESCRIPTION

This fonction read the articular speed of the virtuose.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `speed` corresponds to the articular speed. It is an array of float, with length the number of axes of the virtuose.

VALEUR DE RETOUR

In case of success, the function `virtGetArticularSpeed` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the embedded software is smaller than 3.60

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 101

NOM

virtSetArticularSpeed – send a consigne of articular speed

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetArticularSpeed(VirtContext VC, float* speed);
```

DESCRIPTION

This fonction send a consigne of articular speed to the virtuose.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `speed` corresponds to the articular speed. It is an array of float, with length the number of axes of the virtuose.

VALEUR DE RETOUR

In case of success, the function `virtSetArticularSpeed` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the embedded software is smaller than 3.60

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 102

NOM

virtSetArticularForce – send a consigne of articular force

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetArticularForce(VirtContext VC, float* force);
```

DESCRIPTION

This function send a consigne of articular force to the virtuose. The type of command must be `COMMAND_TYPE_ARTICULAR_IMPEDANCE`.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `force` corresponds to the articular force. It is an array of float, with length the number of axes of the virtuose.

VALEUR DE RETOUR

In case of success, the function `virtSetArticularForce` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the embedded software is smaller than 3.60

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 103

NAME

`virtActiveRotationSpeedControl` – active rotation speed control

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtActiveRotationSpeedControl(VirtContext VC,
                                   float angle,
                                   float speedFactor);
```

DESCRIPTION

The function `virtActiveRotationSpeedControl` activates a hybrid control mode, with a standard position/force control inside a cone, and a simple speed control outside the cone.

Outside the cone, the positions given by `virtGetPosition` and `virtGetAvatarPosition` vary proportionally to the distance to the cone affected by the parameter `speedFactor`, in the same directions, while a constant torque tends to bring back the end-effector of the Virtuose inside the cone.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `angle` corresponds to the angle of the cone.

The parameter `speedFactor` corresponds to a scaling of the rotation speed.

RETURN VALUE

In case of success, the function `virtActiveRotationSpeedControl` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERRORS

`VIRT_E_INVALID_CONTEXT`


The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the embedded software is smaller than 3.60

`VIRT_E_INCORRECT_VALUE`

One of parameters is negative.

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 104

NAME

`virtDeactiveRotationSpeedControl` – disable rotation speed control

SYNOPSIS

```
#include "virtuoseAPI.h"
int virtDeactiveRotationSpeedControl(VirtContext VC);
```

DESCRIPTION

The function `virtDeactiveRotationSpeedControl` leaves the hybrid position/speed control mode, previously activated by `virtActiveRotationSpeedControl`.

PARAMETERS

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

RETURN VALUE

In case of success, the function `virtDeactiveRotationSpeedControl` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERRORS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the embedded software is smaller than 3.60

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 105

NOM

`virtGetControllerVersion` – read the version of the embedded software

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtGetControllerVersion(VirtContext VC,
                           int* major, int* minor);
```

DESCRIPTION

This fonction read the version of the embedded software of the controller.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `major` corresponds to the major index of the version.

The parameter `minor` corresponds to the minor index of the version.


VALEUR DE RETOUR

In case of success, the function `virtGetControllerVersion` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 106

NOM

`virtIsInSpeedControl` – Test whether the operator is in speed control mode

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtIsInSpeedControl(VirtContext VC,
                        int* translation, int* rotation);
```

DESCRIPTION

This function tests whether the operator controls a virtual object in position or in speed mode, in translation and in rotation.

PARAMETRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `translation` corresponds to the translation. A value of 0 indicates the position mode (position of the end-effector inside of the sphere). A value of 1 indicates the speed mode (position of the end-effector outside of the sphere).

The parameter `rotation` corresponds to the rotation. A value of 0 indicates the position mode (orientation of the end-effector inside the cone). A value of 1 indicates the speed mode (orientation of the end-effector outside the cone).

VALEUR DE RETOUR

In case of success, the function `virtIsInSpeedControl` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the embedded software is smaller than 3.60

	VIRTUOSE API V3.60	HAP/02/RT.006
	DOCUMENTATION	Rev. 1 page 107

NOM

virtSetForceInSpeedControl – Set the force in speed control

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetForceInSpeedControl(VirtContext VC,
                               float force);
```

DESCRIPTION

This function set the force in translation when the operator controls a virtual objet in speed control.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `force` corresponds to the force in translation. This force is not saturate, so it is recommend the force to be smaller than the nominal force of the device.

VALEUR DE RETOUR

In case of success, the function `virtSetForceInSpeedControl` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.


ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the embedded software is smaller than 3.60

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 108

NOM

virtSetTorqueInSpeedControl – Set the force in speed control

SYNOPTIQUE

```
#include "virtuoseAPI.h"
int virtSetTorqueInSpeedControl(VirtContext VC,
                                float torque);
```

DESCRIPTION

This function set the torque (in rotation) when the operator controls a virtual objet in speed control.

PARAMÈTRES

The parameter `VC` corresponds to a `VirtContext` object returned by the `virtOpen` function.

The parameter `torque` corresponds to the torque in rotation. This torque is not saturate, so it is recommend the force to be smaller than the nominal torque of the device.

VALEUR DE RETOUR

In case of success, the function `virtSetTorqueInSpeedControl` returns 0. Otherwise, it returns -1 and the function `virtGetErrorCode` gives access to an error code.

ERREURS

`VIRT_E_INVALID_CONTEXT`

The parameter `VC` does not correspond to a valid `VirtContext` object

`VIRT_E_INCOMPATIBLE_VERSION`

The version of the embedded software is smaller than 3.60

6 GLOSSARY

Quaternion

The following formula transforms a rotation of angle θ and vector (a, b, c) into a quaternion:

$$\begin{cases} qx = a \sin\left(\frac{\theta}{2}\right) \\ qy = b \sin\left(\frac{\theta}{2}\right) \\ qz = c \sin\left(\frac{\theta}{2}\right) \\ qw = \cos\left(\frac{\theta}{2}\right) \end{cases}$$

Afterwards, it is possible to normalize the quaternion by dividing each of its four components by its norm, computed as follows:

$$n = \sqrt{qx^2 + qy^2 + qz^2 + qw^2}$$

Cinematic tensor or twist

Considering a solid (S) attached to a reference frame (R1)={O1,x1,y1,z1}, in motion with respect to a fixed reference frame (R0)={O,x0,y0,z0}.

The speed of any point M of (S) with respect to (R0) is given by:


$$\vec{V}(M \in S / R_0) = \left(\frac{dOM}{dt} \right)_{R_0} = \left(\frac{dOO_1}{dt} \right)_{R_0} + x \left(\frac{dx_1}{dt} \right)_{R_0} + y \left(\frac{dy_1}{dt} \right)_{R_0} + z \left(\frac{dz_1}{dt} \right)_{R_0}$$

The label R0 means that, when computing the speed, the base vectors of (R0) are considered constant. We note: $\left(\frac{dOO_1}{dt} \right)_{R_0} = v_x \vec{x}_0 + v_y \vec{y}_0 + v_z \vec{z}_0$

The rotation vector of (S) with respect to (R0) is the vector Ω with the following properties:

$$\begin{cases} \left(\frac{dx_1}{dt} \right)_{R_0} = \vec{\Omega} \wedge \vec{x}_1 \\ \left(\frac{dy_1}{dt} \right)_{R_0} = \vec{\Omega} \wedge \vec{y}_1 \\ \left(\frac{dz_1}{dt} \right)_{R_0} = \vec{\Omega} \wedge \vec{z}_1 \end{cases}$$

The vector Ω does not depend of the choice of (R1) attached to (S).

	VIRTUOSE API V3.60		HAP/02/RT.006
	DOCUMENTATION		Rev. 1 page 110

We note: :

We come to the following formula:

$$V(M \in S / R_0) = V(O_1 / R_0) + \Omega \wedge O_1 M$$

The speed of (S) is the expression of a tensor called cinematic tensor or twist, which reduction components are:

$$(VS / R_0)_{O_1} = \begin{Bmatrix} V_{\Omega}(O_1 \in S / R_0) \\ \Omega(S / R_0) \end{Bmatrix} = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)$$

In this documentation, we call it “speed of S with respect to O1 expressed in R0”.

.